1980

NPS-PASCAL a microcomputer-based implementation of the PASCAL programming language.

Tinius, Konrad Stephen

NPS-PASCAL
A MICROCOMPUTER-BASED IMPLEMENTATION OF THE PASCAL PROGRAMMING LANGUAGE

Konrad Stephen Tinius

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

NPS-PASCAL
A Microcomputer-based Implementation of the
PASCAL Programming Language

by

Konrad Stephen Tinius

March 1980

Thesis Advisor: Bruce J. MacLennan

Approved for public release; distribution unlimited

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) NPS-PASCAL A Microcomputer-based Implementation of the PASCAL Programming Language		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; March 1980
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Konrad Stephen Tinius		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93940		12. REPORT DATE March 1980
		13. NUMBER OF PAGES 219
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Naval Postgraduate School Monterey, CA 93940		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimted

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Microcomputer
Compiler
PASCAL

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

NPS-PASCAL is a student research project at the Naval Postgraduate School, the goal of which is the implementation of the PASCAL programming language on a microcomputer system. NPS-PASCAL will consist of two programs, a compiler which produces intermediate code, and an interpreter,

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE
(Page 1) S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

which will interpret the intermediate code, or a translator, which will produce target machine code. NPS-PASCAL is designed to conform to the requirements of the PASCAL Standard, as defined by the British Standards Institute/International Standards Organization Working Draft/3.

The compiler program, the subject of this thesis, performs the lexical, syntactic and semantic analysis of a PASCAL program. NPS-PASCAL is written in INTEL's PL/M-80 programming language and executes on the CP/M operating system.

Approved for public release; distribution unlimited

NPS-PASCAL
A Microcomputer-based implementation of the
PASCAL programming language

by

Konrad Stephen Tinius
Captain, United States Air Force
B.S., The Ohio State University, 1973

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 1980

ABSTRACT

NPS-PASCAL is a student research project at the Naval Postgraduate School, the goal of which is the implementation of the PASCAL programming language on a microcomputer system. NPS-PASCAL will consist of two programs, a compiler which produces intermediate code, and an interpreter, which will interpret the intermediate code, or a translator, which will produce target machine code. NPS-PASCAL is designed to conform to the requirements of the PASCAL Standard, as defined by the British Standards Institute/International Standards Organization Working Draft/3.

The compiler program, the subject of this thesis, performs the lexical, syntactic and semantic analysis of a PASCAL program. NPS-PASCAL is written in INTEL's PL/M-80 programming language and executes on the CP/M operating system.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

NPS-PASCAL is an implementation of the PASCAL programming language on a microcomputer system. NPS-PASCAL is a continuing research project in the Computer Science Department at the Naval Postgraduate School, Monterey, California. The original NPS-PASCAL design and programs were written by MAJ Joaquin C. Gracida, USMC, and LT Robert R. Stilwell (SC) USN, in their thesis submitted June 1978. Their work is contained in Ref. 1. MAJ Gracida and LT Stilwell implemented the basic contructs of the PASCAL language in a one-pass compiler and code generator. Thesis work was continued in June 1979 by LT John L. Byrnes, USN, who added code to implement many missing constructs, and developed a number of user assistance programs. His work is contained in Ref. 2. Thesis work was continued again in October 1979, with the goal of completing the compiler portion of NPS-PASCAL. Follow-on thesis work will lead to an NPS-PASCAL interpreter/translator and a complete PASCAL system. In the discussion which follows, it is assumed that the reader is familiar with Refs. 1 and 2.

B. APPROACH

The first step in continuing the NPS-PASCAL project was to convert the source programs from PL/M to PL/M-80 and transfer them from the IBM 360/67-based timesharing system to the Intel Microprocessor Development System. This would

permit the compiler to be developed and debugged in a completely microprocessor oriented environment, and would eliminate the need to use the PL/M cross compiler.

The next step was to study the program listings and previous theses to gain a detailed familiarity with the project. Included in this step was acquiring a working knowledge of the Intel ISIS-II operating system and the PL/M-80 compiler and its attendant linking and loading programs and utilities. Since NPS-PASCAL is compiled under the ISIS-II operating system, but executes under the CP/M operating sysem, it was also necessary to learn the CP/M utilities for transferring files between systems, and the CP/M run-time debuggers, DDT and SID.

The largest portion of this thesis effort consisted of making corrections and additions to existing code, adding code where necessary, tracing execution to locate logic and data errors, correcting documentation, and running test PASCAL programs. Implementation of the record construct required changing the original grammar and correcting the parse tables.

To avoid testing the compiler with syntactically incorrect PASCAL programs, test programs were selected from the <u>PASCAL User Manual and Report</u> [3], from various student texts on PASCAL, and from the PASCAL Validation Suite[4]. The test programs from the Validation Suite were particularly helpful, in that they exercised the full range of any given PASCAL construct.

An attempt was made to upgrade and complete the SYMBOLTABLE user assistance program described in Ref. 2, however, it was abandoned and a substitute program, SYMDUMP, was developed. SYMDUMP provides an ordered, addressed hex dump of the symbol table, and provides a much more useful and efficient means of accessing the symbol table.

It was felt that it would be beneficial to include and consolidate the documentation and descriptions from the previous theses into a single document, so sections of Refs. 1 and 2 appear in this thesis. The appropriate sections were updated to reflect changes in the program code or structure. In others, descriptions were expanded and diagrams were added, or the section was included in its entirety.

II. NPS-PASCAL COMPILER IMPLEMENTATION

A. NPS-PASCAL LANGUAGE BACKGROUND

NPS-PASCAL is an implementation of the PASCAL language based on the BSI/ISO Working Draft/3 of Standard Pascal [5], referred to in this thesis as "STANDARD PASCAL." NPS-PASCAL is in compliance with STANDARD PASCAL's definition of a conforming processor with the following three exceptions:

(1) Identifiers, directives, and labels can be of any length, as prescribed by STANDARD PASCAL, provided their uniqueness can be determined from the first thirty characters.

(2) Integers are limited to any value between -32,768 and +32,767. Real values can take on any negative or positive value consisting of fourteen digits multiplied by ten to the -64th power through ten to the +63rd power.

(3) "EOP" is a special symbol, or reserved word, in the NPS-PASCAL vocabulary indicating "end of program."

Consequently, any program that conforms to the rules of STANDARD PASCAL, and meets the above listed qualifications, constitutes a syntactically correct NPS-PASCAL program.

The University of Toronto's parse table generator [6] was used to specify the NPS-PASCAL grammar in LALR(1) form. The generator operates on the IBM 360/67 and produces parse tables for the language, thus permitting extensions and corrections to be made in an easy and efficient manner.

B. COMPILER ORGANIZATION

The compiler structure, diagrammed in Fig. 1, performs a single pass through the source program, produces an intermediate language file and may print an optional listing of the source program to the console. The one pass approach was taken to provide speed and to reduce the size of the compiler. The disadvantage of the one-pass design is the inability to specify the exact location where program execution resumes after a forward branch. To solve this problem, labels are placed in the intermediate code where execution should continue. The resolution of label locations is then the responsibility of the interpreter/translator as it scans the intermediate code.

The compiler builds the symbol table, converts all numbers to their internal representation, and generates the intermediate code file and the symbol table file. The compiler accepts input parameters to control the listing of the source program, production numbers, or token numbers. The creation of the intermediate file can also be suppressed if it is not needed.

C. SCANNER

The scanner analyzes the source program character by character and passes each token identified to the parser. The scanner can provide a listing of the source statements and eliminate comments.

The scanner is written in four sections which are selectively executed depending on the first non-blank

NPS-PASCAL Compiler Structure

```
                          --------------
                          |            |
                          |  TEST.PAS  |
                          |            |
                          --------------
                                 |
                                 |
       ------------          -----------
       | RESERVED |         /           \
       |   WORD   |-----|   SCANNER     |
       |  TABLE   |         \           /
       ------------          -----------
                                 |
       ------------              |
     /             \             |
    | INITIALIZER   |            |
     \             / \           |
       ------------   \          |
                       \         |
         -----------    \        |
         |  PARSE  |          -----------        ----------
         | TABLES  |------| PARSER    |-------| PARSE    |
         -----------          -----------        | STACKS   |
                         /       |               ----------
        ------------   /         |              /
      /    ERROR     \/          |             /
      \   HANDLER    /           |            /
        ------------             |           /
                                 |          /
                          -------------
                          | INT CODE  |
                          | GENERATOR |
                          -------------
                                 |
                                 |
                          --------------
                          |  TEST.PIN  |
                          --------------
```

FIGURE 1.

character of the input string. When the section to execute has been determined, the remainder of the token is scanned and placed in the input array ACCUM. The first byte of the ACCUM array contains the length of the token. In the case of tokens that exceed the size of the ARRAY (32 bytes), a continuation flag is set to allow the scanner and parser to accept the rest of the token.

The four sections of the scanner process strings, numbers, identifiers and reserved words, and special characters, respectively. The string processing section is executed whenever the first character of the token is a quotation mark. The scanner then accepts each succeeding character until a second quotation mark is found, indicating the end of the string. The section that processes numbers determines the type of the number being scanned as it scans each character. This determination is used by subroutines later in the compilation process to perform type checking and conversion to internal representation. When the scanner recognizes an identifier, it searches the vocabulary table to determine if it is a reserved word. If so, the scanner returns the token number associated with the reserved word. Special characters found in the vocabulary table are handled as separate tokens except in two cases. If a period is followed immediately by numeric characters, the scanner assumes a real number is being scanned. When a pair of special characters occurs consecutively, (for instance :=), the scanner passes both characters as a single token after

assigning the appropriate token number from the vocabulary table.

D. SYMBOL TABLE

The symbol table is used to store the attributes of labels, constants, type declarations, variable identifiers, procedures, functions and file declarations. This stored information is used by the compiler to verify that the program is semantically correct and to assist in code generation. Access to the symbol table is through various subroutines using based global variables to uniquely address the elements of each entry.

1. Symbol Table Construction.

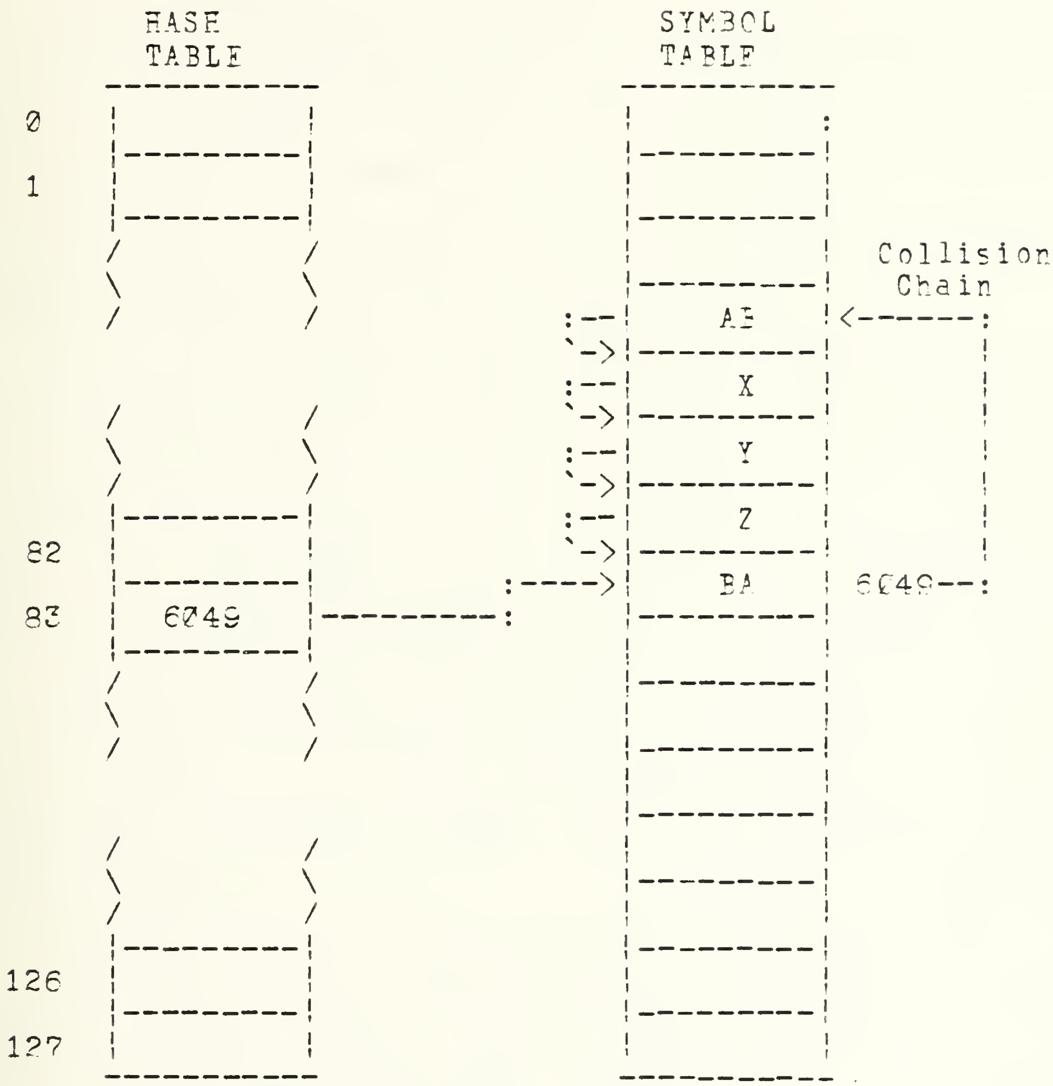
The symbol table is an unordered linked list of entries which grows from the last byte of the compiler toward high memory. Individual entries are either accessed via a chained hash addressing technique (as illustrated in Figure 2), or by means of address pointer fields contained in other entries. This latter method of access is required since not all entries in the symbol table have an identifier, called a printname, associated with them.

Each location in the hash table contains the head of a singly linked list of entries whose printname, when evaluated, results in the same hash value. A zero in any cell of the hash table indicates that there are no entries whose printname produces that value. During symbol table construction or access, the global variable PRINTNAME contains the address of a string of bytes whose first

HASHING FUNCTION: SUM OF THE VALUES OF THE ASCII CHARACTERS OF THE PRINTNAME MODULO 128.

EXAMPLE:

Identifier AB = (41H + 42H) MOD 80H = 83H
Identifier BA = (41H + 41H) MOD 80H = 83H



SYMBOL TABLE ACCESS

FIGURE 2.

element is the length of the current identifier, followed by the identifier's ASCII characters. The global variable SYMHASH contains the hash code value of the identifier. The hash code is the sum of the hex values of the PRINTNAME's ASCII characters, modulo 128 (base 10). Entries that produce the same hash code are linked together in the symbol table by a chain which is accessed via the entry's collision field. The chain is constructed in such a way as to have the most recent entry at the head of the chain.

Each entry in the symbol table contains a number of fields, some of which are common to all entries, and some of which apply only to particular classes of entries. All entries have the same first three fields: the collision field in the first two bytes; the previous symbol table entry address field in the third and fourth bytes; and the form field in the fifth byte. The remaining fields are used to uniquely describe each entry's attributes and characteristics.

There are eight different types of entries in the NPS-PASCAL symbol table. Each of these types has a unique three bit code in the right-most three bits of its form field. The remaining five bits in the form field furthur subdivide the entry types among the eight classes according to the particular characteristics of the type involved. The form field bit assignments are summarized in Table 1. The characteristics are described in detail as each type of symbol table entry is presented below.

FORM Field Organization

Form Value	Name of Entry	Bit Pattern
00H	Label	00 000 000
x1H	Constant	
01H	Unsigned identifier	00 000 001
41H	Signed identifier	01 000 001
09H	Integer	00 001 001
11H	Real	00 010 001
19H	String	00 011 001
x2H	Type	
42H	Integer	01 000 010
4AH	Real	01 001 010
52H	Char	01 010 010
5AH	Boolean	01 011 010
7AH	Type declaration	01 111 010
x3H	Variable	
03H	Scalar	00 000 011
0BH	Integer	00 001 011
13H	Character	00 010 011
1BH	Real	00 011 011
23H	Complex	00 100 011
2BH	Boolean	00 101 011
x4H	Procedure	
04H	Procedure	00 000 100
x5H	Function	
05H	Function	00 000 101
x6H	File	
06H	File	00 000 110
x7H	User defined	
07H	Scalar	00 000 111
0FH	Enumerated subrange	00 001 111
4FH	Integer subrange	01 001 111
8FH	Character subrange	10 001 111
17H	Array	00 010 111
1FH	Record	00 011 111
5FH	Field (of record)	01 011 111
9FH	Tag field	10 011 111
DFH	Variant field	11 011 111
27H	Set	00 100 111
2FH	File	00 101 111
37H	Pointer	00 110 111

Table 1.

a. Label entries

The form field of a label entry has the value of 00H. The hash value of the label's printname is in the next byte; the hash value is stored for collision resolution later. The length of the label follows in the next one byte field. The printname characters appear, one per byte, after the length field. A two byte field following the printname characters contains a sequentially generated integer value which is assigned as the label's internal label number. This value is used as the target for branching in the intermediate code. An example of a label entry is shown in Fig. 3.

b. Constant Entries

The form field of a constant symbol table entry identifies the type of entry, and the particular type of the constant as well. There are five valid types of constants in NPS-PASCAL: an unsigned identifier with FORM = 01H; a signed identifier with FORM = 41H; an integer with FORM = 09H; a real value with FORM = 11H; and a string constant with FORM = 19H. Following the form field are the printname hash field, the length field, and the printname characters.

The value field may consist of another length field and the printname characters in the case of identifier and string constants, or it may contain the internal representation of a constant number (two bytes for integers or eight bytes for reals). Two examples of constant entries are shown in Figs. 4 and 5.

LABEL 67;

Memory Address	Symbol Table	
73ØØH	A5H	\ COLLISION
		>
73Ø1H	Ø1H	/ ADDRESS
73Ø2H	3DH	\ PREVIOUS SBTBL
		>
73Ø3H	Ø3H	/ ENTRY ADDRESS
73Ø4H	ØØH	FORM
73Ø5H	6DH	BASE
73Ø6H	Ø2H	PRINTNAME LENGTH
73Ø7H	36H	ASCII CHARACTER 6
73Ø8H	37H	ASCII CHARACTER 7
73Ø9H	ØØH	\ LABEL NUMBER
		>
73ØAH	ØØH	/ "Ø"

SYMBOL TABLE LABEL ENTRY

FIGURE 3.

CONST BOIL = 212;

Memory Address	Symbol Table	
730BH	00H	COLLISION
730CH	00H	ADDRESS
730DH	00H	PREVIOUS SBTBL
730EH	73H	ENTRY ADDRESS
730FH	09H	FORM
7310H	26H	HASH
7311H	04H	PRINTNAME LENGTH
7312H	42H	ASCII CHARACTER B
7313H	4FH	ASCII CHARACTER O
7314H	49H	ASCII CHARACTER I
7315H	4CH	ASCII CHARACTER L
7316H	D4H	CONSTANT VALUE
7317H	00H	OF ENTRY

SYMBOL TABLE UNSIGNED INTEGER

CONSTANT ENTRY

FIGURE 4.

CONST BOIL = 'BOIL';

Memory Address	Symbol Table	
730BH	00H	\ COLLISION
		>
730CH	00H	/ ADDRESS
730DH	00H	\ PREVIOUS SBTBL
		>
730EH	73H	/ ENTRY ADDRESS
730FH	19H	FORM
7310H	26H	HASH
7311H	04H	PRINTNAME LENGTH
7312H	42H	ASCII CHARACTER B
7313H	4FH	ASCII CHARACTER O
7314H	49H	ASCII CHARACTER I
7315H	4CH	ASCII CHARACTER L
7316H	04H	STRING LENGTH
7317H	42H	ASCII CHARACTER B
7318H	4FH	ASCII CHARACTER O
7319H	49H	ASCII CHARACTER I
731AH	4CH	ASCII CHARACTER L

SYMBOL TABLE STRING CONSTANT ENTRY

FIGURE 5.

c. Type entries

NPS-PASCAL has two kinds of type entries in its symbol table: simple type entries and type declaration entries. The simple type entry can either be one of NPS-PASCAL's standard types, or a previously defined simple type declaration (scalar or subrange). In the latter case, a simple type entry is made in the symbol table, with a pointer to the scalar or subrange type declaration entry. In the former case, one of the following standard types will be assigned to the type entry.

Integer - The values of this type are a subset of the whole numbers whose range is the set of values: -maxint,-maxint+1,...,-1,0,1,...maxint-1,maxint where maxint = 32,767.

Real - The values are a subset of the real numbers consisting of fourteen digits multiplied by ten to the -64th power through ten to the +63rd power.

Boolean - The values are denoted by the identifiers "false" and "true", such that false is less than true.

Character - The values of this type are the defined set of characters described in Ref 5. The following relationships hold for character types:

(1) The subset of character values representing the digits 0 through 9 is ordered and contiguous.

(2) The subset of character values representing the upper case letters A through Z is ordered and contiguous.

(3) The subset of character values representing the lower case letters a through z is ordered and contiguous.

Type declarations entries, however, are generated from user defined types found elsewhere in the source program. It is possible to define a chain of type declarations. An example would be an array of the type array which is itself of type integer.

The symbol table entry for a type is as follows. An integer type has the FORM value of 42H, a real type has the FORM value of 4AH, a character type has the FORM value of 52H, and a boolean type has the FORM value of 5AH. A FORM value of 7AH indicates that an additional type declaration entry must be accessed. The field following the form is a one byte field containing the hash value of the printname. The next byte contains the printname's length, which is followed by the printname characters of the type identifier. The last two bytes contain the address of the specified type. Examples of simple type entries are shown in Figs. 6 - 9.

TYPE NUM = INTEGER;

Memory Address	Symbol Table	
7322H	00H	COLLISION
7323H	00H	ADDRESS
7324H	18H	PREVIOUS SBTBL
7325H	73H	ENTRY ADDRESS
7326H	42H	FORM
7327H	70H	HASH
7328H	03H	PRINTNAME LENGTH
7329H	4EH	ASCII CHARACTER N
732AH	55H	ASCII CHARACTER U
732BH	4DH	ASCII CHARACTER M
732CH	06H	SBTBL ADDRESS OF
732DH	01H	PARENT TYPE

SYMBOL TABLE SIMPLE TYPE INTEGER ENTRY

FIGURE 6.

TYPE NUM = CHAR;

Memory Address	Symbol Table	
7322H	00H	COLLISION
7323H	00H	ADDRESS
7324H	18H	PREVIOUS SBTBL
7325H	73H	ENTRY ADDRESS
7326H	52H	FORM
7327H	70H	HASH
7328H	03H	PRINTNAME LENGTH
7329H	4EH	ASCII CHARACTER N
732AH	55H	ASCII CHARACTER U
732BH	4DH	ASCII CHARACTER M
732CH	1FH	SBTBL ADDRESS OF
732DH	01H	PARENT TYPE

SYMBOL TABLE SIMPLE TYPE ENTRY

FIGURE 7.

TYPE NUM = BOOLEAN;

Memory Address	Symbol Table	
7322H	ØØH	\ COLLISION
		>
7323H	ØØH	/ ADDRESS
7324H	18H	\ PREVIOUS SBTBL
		>
7325H	73H	/ ENTRY ADDRESS
7326H	5AH	FORM
7327H	7ØH	HASH
7328H	Ø3H	PRINTNAME LENGTH
7329H	4EH	ASCII CHARACTER N
732AH	55H	ASCII CHARACTER U
732BH	4DH	ASCII CHARACTER M
732CH	2AH	SBTBL ADDRESS OF
732DH	Ø1H	PARENT TYPE

SYMBOL TABLE SIMPLE TYPE ENTRY

FIGURE 8.

TYPE NUM = REAL;

Memory Address	Symbol Table	
7322H	00H	\ COLLISION
		>
7323H	00H	/ ADDRESS
7324H	18H	\ PREVIOUS SBTBL
		>
7325H	73H	/ ENTRY ADDRESS
7326H	4AH	FORM
7327H	70H	HASH
7328H	03H	PRINTNAME LENGTH
7329H	4EH	ASCII CHARACTER N
732AH	55H	ASCII CHARACTER U
732BH	4DH	ASCII CHARACTER M
732CH	14H	SBTBL ADDRESS OF
732DH	01H	PARENT TYPE

SYMBOL TABLE SIMPLE TYPE ENTRY

FIGURE 9.

There are seven different user definable types in NPS-PASCAL. A type declaration entry is constructed whenever a scalar type, subrange type, array type, record type, set type, file type, or pointer type is encountered.

(1) <u>Scalar Types</u>. By definition, a scalar type is an ordered set of values whose identifiers are enumerated to denote their values. The form field entry for scalar types has the value 07H. Scalar entries are the only type declaration entries that have an accessible printname. Consequently, the next two fields hold the printname hash value and length. The printname characters follow these fields. The next field is a byte value containing the enumerated value of the scalar identifier. The enumerated values (0,1,2...) are assigned to the scalars in the order in which they appear in the declaration. The final field is a two byte field storing the symbol table address of the parent type. The scalar type entry will be pointed to by the variable entry claiming this type. An example of a scalar type entry is presented in Fig. 10.

(2) <u>Subrange Types</u>. A subrange type is a duplicate declaration of any other previously defined scalar type, integer type, or character type, but with a specified lower and upper bound on its elements. The form field of a subrange entry is 0FH for enumerated elements, 4FH for integer elements, and 8FH for character elements. Bytes six and seven store the address of the subrange element's parent type. Bytes eight and nine hold the low value of the range,

Memory Address	Symbol Table	
733CH	ØØH	\ COLLISION
733DH	ØØH	/ ADDRESS
733EH	2FH	\ PREVIOUS SBTBL
733FH	73H	/ ENTRY ADDRESS
734ØH	Ø7H	FORM
7341H	5BH	HASH
7342H	Ø3H	PRINTNAME LENGTH
7343H	52H	ASCII CHARACTER R
7344H	45H	ASCII CHARACTER E
7345H	44H	ASCII CHARACTER D
7346H	ØØH	ENUMERATION VALUE
7347H	2FH	\ SBTBL ADDRESS OF
7348H	73H	/ PARENT TYPE

SYMBOL TABLE SIMPLE TYPE ENTRY (SCALAR)

FIGURE 1Ø.

while the next two bytes contain the high value of the range. The following field is two bytes long and stores the total number of elements in the range. The displacement vector is not stored with the subrange, since any given subrange could serve as the index to arrays of different base types. The displacement vector is stored instead with the array entry itself. This entry will be pointed to by a variable entry claiming this type. An example of a subrange type entry is shown in Fig. 11.

(3) Array Types. The preceding two type declaration entries in NPS-PASCAL are called simple type entries. They are symbol table entries using a single, predefined type. Structured types are compositions of types. In other words, one or more types are used to describe a single symbol table entry. A structured type will have a type declaration entry which contains the printname, and which points to the structure type entry.

The array type is a structured type consisting of a fixed number of components that are all of the same type, called the component type. The number of components is specified as a scalar or subrange type and is referred to as the index type. INTEGER and REAL types are not legal index types; however, the scalar or subrange type can be of the type integer.

The symbol table format for an array entry has the form value of 17H. The following byte specifies the number of indices, or dimensions in the array. The next two

TYPE PRIME = RED..BLUE;

Memory Address	Symbol Table	
73A2H	00H	\ COLLISION
73A3H	00H	/ ADDRESS
73A4H	94H	\ PREVIOUS SBTBL
73A5H	73H	/ ENTRY ADDRESS
73A6H	0FH	FORM
73A7H	2FH	\ SBTBL ADDRESS OF
73A8H	73H	/ PARENT TYPE
73A9H	00H	\ SUBRANGE
73AAH	00H	/ LOW VALUE
73ABH	02H	\ SUBRANGE
73ACH	00H	/ HIGH VALUE
73ADH	03H	NUMBER OF ELEMENTS

SYMBOL TABLE SUBRANGE TYPE ENTRY

FIGURE 11.

fields are both two bytes long, the first containing the address of the component type; the second containing the total storage requirement for the array in bytes. The eleventh byte of the entry holds a value designating the type of the array's component as defined in Table 2. A two byte field follows with the symbol table address of the type entry of the array's first dimension. This is followed by a two byte field which contains the displacement vector for this dimension. The displacement vector for each dimension represents the distance in bytes between two elements of the array which have a difference of one in the corresponding subscript. If the array has more than one dimension, four more bytes are alloted in the symbol table to store the address and displacement vector of each additional dimension. This entry will be pointed to by the variable entry claiming this type. An example of an array type entry is shown in Fig. 12.

(4) Record Types. A record is another NPS-PASCAL structured type. This structure has a fixed number of components, called fields, each of which can be of any defined type. The symbol table entry for a record has the form field value of 1FH. Bytes six and seven contain the storage requirements in bytes for the entire record. Bytes eight and nine store the symbol table address of the type entry of the last field contained in the record structure. The remaining field entries are located by chaining backward to the parent record entry via the previous symbol table

BASIC TYPE OF COMPONENTS

Value	Meaning (Type)
00H	Ordinate
01H	Integer
02H	Character
03H	Real
04H	Complex
05H	Boolean

TABLE 2.

TYPE MIX ARRAY[1..6] OF COLOR;

Memory Address	Symbol Table	
73CCH	00H	\ COLLISION
73CDH	00H	/ ADDRESS
73CEH	BDH	\ PREVIOUS SBTBL
73CFH	73H	/ ENTRY ADDRESS
73D0H	17H	FORM
73D1H	01H	NUMBER OF DIMENSIONS
73D2H	2EH	\ SBTBL ADDRESS OF
73D3H	73H	/ COMPONENT TYPE
73D4H	06H	\ TOTAL STORAGE REQUIRED
73D5H	00H	/ IN BYTES
73D6H	00H	TYPE OF COMPONENT
73D7H	01H	\ ARRAY
73D8H	00H	/ OFFSET
73D9H	EDH	\ SBTBL ADDRESS OF
73DAH	73H	/ FIRST DIMENSION
73DBH	FFH	\ DISPLACEMENT VECTOR
73DCH	03H	/ OF FIRST DIMENSION

SYMBOL TABLE ARRAY TYPE ENTRY

FIGURE 12.

entry address. An example of a record type entry is shown in Fig. 13.

Each record field consists of an identifier and a type. The form field of a record entry has a value of 5FH. The following two fields are bytes for the hash and the length of the printname. The next field holds the printname characters. The address of the parent record is stored in the next two bytes. The following field has a one byte length and is used to store the record field's type. The value stored is also taken from Table 2. Two more bytes are used to store the symbol table address of the type just indicated. The last field of this entry is two bytes long and holds the offset of the record field from the record base.

NPS-PASCAL supports the variant field and tag field constructs of records. These two kinds of record fields have symbol table entries similar to the one described above for fields, with the exception of the form field, which is DFH for variant fields, and 9FH for tag fields. An example of a field entry is shown in Fig. 14.

(5) Set Types. The set structure defines a set of values which is the power set of a declared base type. The base type is required to be a scalar or subrange type. The set type symbol table entry has a form field value of 27H. The following two bytes contain the symbol table address of the set type identifier. An example of a set type entry is shown in Fig. 15.

```
TYPE COMPLEX=RECORD
          RE,IM:REAL;
          END;
```

Memory Address	Symbol Table	
7Ø8BH	ØØH	COLLISION ADDRESS
7Ø8CH	ØØH	
7Ø8DH	7BH	PREVIOUS SBTBL ENTRY ADDRESS
7Ø8EH	7ØH	
7Ø8FH	1FH	FORM
7Ø9ØH	1ØH	TOTAL STORAGE REQUIRED IN BYTES
7Ø91H	ØØH	
7Ø92H	A4H	SBTBL ADDRESS OF LAST RECORD FIELD
7Ø93H	7ØH	

SYMBOL TABLE RECORD TYPE ENTRY

FIGURE 13.

```
TYPE COMPLEX=RECORD
          RE,IM:REAL;
          END;
```

Memory Address	Symbol Table	
7Ø94H	A8H	\ COLLISION
7Ø95H	Ø2H	/ ADDRESS
7Ø96H	8FH	\ PREVIOUS SBTBL
7Ø97H	7ØH	/ ENTRY ADDRESS
7Ø98H	5FH	FORM
7Ø99H	17H	HASH
7Ø9AH	Ø2H	PRINTNAME LENGTH
7Ø9BH	52H	ASCII CHARACTER R
7Ø9CH	45H	ASCII CHARACTER E
7Ø9DH	8BH	\ SBTBL ADDRESS OF
7Ø9EH	7ØH	/ PARENT RECORD
7Ø9FH	Ø3H	TYPE OF THIS FIELD
7ØAØH	14H	\ SBTBL ADDRESS OF
7ØA1H	Ø1H	/ FIELD TYPE
7ØA2H	ØØH	\ OFFSET FROM
7ØA3H	ØØH	/ RECORD BASE

SYMBOL TABLE RECORD FIELD ENTRY

FIGURE 14.

TYPE FLAG=SET OF COLOR;

Memory Address	Symbol Table	
73E8H	ØØH	COLLISION
73E9H	ØØH	ADDRESS
73EAH	DBH	PREVIOUS SBTBL
73EBH	73H	ENTRY ADDRESS
73ECH	27H	FORM
73EDH	2FH	SBTBL ADDRESS OF
73EEH	73H	SET TYPE

SYMBOL TABLE SET TYPE ENTRY

FIGURE 15.

(6) File Types. A NPS-PASCAL structure consisting of a sequence of components, all of the same type, is called a file. A file type indicates a natural ordering of the components, whose position in the file defines the sequence. A file type declaration entry in the symbol table has a form field value of 2FH. The symbol table address of the file type's identifier is contained in the next two bytes. An example of a file type entry is shown in Fig. 16.

(7) Pointer Types. NPS-PASCAL supports dynamic variables which are generated without any correlation to the static structure of the program. These variables are assigned a special type called pointer type. The form field value is set to 37H, while bytes six and seven hold the symbol table address of the pointer type's parent entry. An example of a pointer type entry is shown in Fig. 17.

d. Variable Entries

Each variable declared in an NPS-PASCAL program is inserted into the symbol table. The form field of the variable entry contains a value which describes the type of the variable. The values for this field and the associated types are shown in Table 1. Following the form field are the fields containing the variable identifier's printname, hash value, length, and the printname characters. A two byte field which contains the variable's starting address in memory appears after the printname characters. This address is an offset from the base of the variable area, called the

TYPE DATA=FILE OF NUM;

Memory Address	Symbol Table	
73FEH	00H	\ COLLISION
73FFH	00H	/ ADDRESS
7400H	EFH	\ PREVIOUS SBTBL
7401H	73H	/ ENTRY ADDRESS
7402H	2FH	FORM
7403H	22H	\ SBTBL ADDRESS OF
7404H	73H	/ FILE TYPE
7400H	00H	ASCII CHARACTER Z

SYMBOL TABLE FILE TYPE ENTRY

FIGURE 16.

TYPE P:^PRIME;

Memory Address	Symbol Table	
7430H	00H	\ COLLISION
7431H	00H	/ ADDRESS
7432H	21H	\ PREVIOUS SBTBL
7433H	74H	/ ENTRY ADDRESS
7434H	37H	FORM
7435H	94H	\ SBTBL ADDRESS OF
7436H	73H	/ POINTER TYPE

SYMBOL TABLE POINTER TYPE ENTRY

FIGURE 17.

Program Reference Table (PRT), which address is assigned by the NPS-PASCAL code generator. The variable's type determines the number of bytes assigned to store the variable in the PRT. The compiler keeps a running total of the amount of storage assigned to all variables, and includes this value in the pseudo code at the completion of a successful program compilation. The interpreter/translator subsequently converts the relative addresses in the intermediate code to absolute address in the final target machine. Next is a two byte field which contains the SBTBL address of the variable's type. In the case of the standard Pascal types integer (FORM = 0BH), real (1BH), character (13H) and boolean (23H), this is the address of that type in the BUILTINTBL. In the case of integer and character subranges (27H), this field contains the address of the subrange type entry. In the case of a scalar (03H), this field contains the address of the last of a series of scalar (07H) entries. The remaining scalar entries are located by chaining backward to the variable entry via the previous symbol table entry address. If the variable is a complex declaration, (array, record, set, file or pointer), this field contains the address of the complex type's entry in the symbol table. If the variable is of a type previously defined in the program, this field contains a pointer to that type declaration. Examples of variable entries are shown in Figs. 18 - 20.

VAR X:INTEGER;

Memory Address	Symbol Table	
6FCBH	ØØH	COLLISION
6FCCH	ØØH	ADDRESS
6FCDH	3DH	PREVIOUS SBTBL
6FCEH	Ø3H	ENTRY ADDRESS
6FCFH	ØBH	FORM
6FDØH	58H	BASE
6FD1H	Ø1H	PRINTNAME LENGTH
6FD2H	58H	ASCII CHARACTER X
6FD3H	3ØH	PRT LOCATION
6FD4H	ØØH	ASSIGNED
6FD5H	Ø6H	SYMBOL TABLE ADDRESS
6FD6H	21H	OF PARENT TYPE

SYMBOL TABLE VARIABLE ENTRY (INTEGER)

FIGURE 18.

VAR OP:(PLUS,MINUS,TIMES);

```
 Memory    |   Symbol    |
 Address   |   Table     |
           |-------------|
  7014H    |    00H      |   \   COLLISION
           |-------------|    >
  7015H    |    00H      |   /   ADDRESS
           |-------------|
  7016H    |    07H      |   \   PREVIOUS SBTBL
           |-------------|    >
  7017H    |    70H      |   /   ENTRY ADDRESS
           |-------------|
  7018H    |    03H      |   FORM
           |-------------|
  7019H    |    1FH      |   HASH
           |-------------|
  701AH    |    02H      |   PRINTNAME LENGTH
           |-------------|
  701BH    |    4FH      |   ASCII CHARACTER O
           |-------------|
  701CH    |    50H      |   ASCII CHARACTER P
           |-------------|
  701DH    |    0FH      |   \   PRT LOCATION
           |-------------|    >
  701EH    |    00H      |   /   ASSIGNED
           |-------------|
  701FH    |    3EH      |   \   SBTBL ADDRESS OF
           |-------------|    >
  7020H    |    70H      |   /   LAST SCALAR
           |-------------|
           |             |
           |-------------|
           |             |
           |-------------|
           |             |
           |-------------|
           |             |
           |-------------|
           |             |
           |-------------|
           |             |
           |-------------|
           |             |
```

SYMBOL TABLE VARIABLE ENTRY (SCALAR)

FIGURE 19.

VAR A:ARRAY[1..5] OF INTEGER;

Memory Address	Symbol Table	
7721H	00H	\ COLLISION
7722H	00H	/ ADDRESS
7723H	14H	\ PREVIOUS SBTBL
7724H	77H	/ ENTRY ADDRESS
7725H	23H	FORM
7726H	41H	HASH
7727H	01H	PRINTNAME LENGTH
7728H	41H	ASCII CHARACTER A
7729H	03H	\ PPT LOCATION
772AH	00H	/ ASSIGNED
772BH	3AH	\ SBTBL ADDRESS
772CH	77H	/ OF ARRAY TYPE ENTRY

SYMBOL TABLE VARIABLE ENTRY (COMPLEX)

FIGURE 20.

e. Procedure and Function Entries

Every procedure and function in an NPS-PASCAL program has an associated entry in the symbol table. In the case of a procedure entry, the form field is assigned the value 04H. The hash value, length of the printname, and the printname characters immediately follow the form field. A one byte field follows and stores the number of parameters associated with the procedure. A two byte field is next, storing the symbol table location of a listing of the procedure's parameter types. This listing is referenced by the compiler to ensure proper mapping, and is located immediately after the final procedure entry in the symbol table. Following the parameter type's address field in the procedure entry are three more two byte fields. The first field gives the PRT address assigned to the procedure identifier. The second field gives the PRT address assigned to the procedure save block pointer (SBP). The SBP permits recursive subroutine calls, and will be explained in the section on Code Generation. The final field in the entry holds a label value that must be branched to when the procedure is invoked. An example of a procedure entry is shown in Fig. 21.

A function entry in the symbol table duplicates a procedure entry with two exceptions. A function entry has a form field value of 05H; and one byte field is added at the end of the entry to designate the type of the function.

PROCEDURE LO (X:INTEGER; VAR Y:INTEGER);

Memory Address	Symbol Table	
746DH	00H	COLLISION
746EH	0CH	ADDRESS
746FH	60H	PREVIOUS SBTBL
7470H	74H	ENTRY ADDRESS
7471H	04H	FORM
7472H	1BH	HASH
7473H	02H	PRINTNAME LENGTH
7474H	4CH	ASCII CHARACTER L
7475H	4FH	ASCII CHARACTER O
7476H	02H	NUMBER OF PARAMETERS
7477H	97H	SBTBL ADDRESS OF
7478H	74H	PARAMETER LISTING
7479H	22H	PRT LOCATION
747AH	00H	ASSIGNED
747BH	28H	SAVE BLOCK POINTER
747CH	28H	LOCATION IN PRT
747DH	01H	LABEL PRECEEDING
747EH	00H	PROCEDURE CODE

SYMBOL TABLE PROCEDURE ENTRY

FIGURE 21.

Function type assignments are also taken from Table 2. An example of a function entry is shown in Fig. 22.

(1) Formal Parameters. Formal parameters provide a mechanism that allows a procedure or function to be repeated with various values being substituted. The formal parameters are declared in the procedure or function declaration and can be of four types: value parameters, variable parameters, procedure parameters and function parameters. Each declared parameter has an associated symbol table entry. A value parameter entry has exactly the same format as the variable entry. A variable parameter entry also duplicates a variable symbol table entry, with the exception of the form field. The high order bit of the form field is set to one for all variable parameters. Procedure and function parameters are entered as described above for procedure and function symbol table entries.

Figure 23 illustrates a sample series of symbol table entries with a procedure entry followed by various formal parameter entries. Note that the final few bytes show the listing of the procedure's parameter types that will be utilized for mapping actual parameters into the formal parameters.

E. PARSER

The parser is a table driven automaton and is modelled after the ALGOL-M [7]. The LALR(k) parser generator [6] produced the required parse tables and the vocabulary table, VOCAB. The parser operates by receiving tokens from the

FUNCTION YZ(F,A:REAL):REAL;

Memory Address	Symbol Table	
746DH	00H	\ COLLISION
746EH	00H	/ ADDRESS
746FH	60H	\ PREVIOUS SBTBL
7470H	74H	/ ENTRY ADDRESS
7471H	05H	FORM
7471H	33H	HASH
7473H	02H	PRINTNAME LENGTH
7474H	59H	ASCII CHARACTER Y
7475H	5AH	ASCII CHARACTER Z
7476H	02H	NUMBER OF PARAMETERS
7477H	CAH	\ SBTBL ADDRESS OF
7478H	74H	/ PARAMETER LISTING
7479H	22H	\ PRT LOCATION
747AH	00H	/ ASSIGNED YZ
747BH	46H	\ SAVE BLOCK POINTER
747CH	00H	/ LOCATION IN PRT
747DH	07H	\ LABEL PRECEEDING
747EH	00H	/ PROCEDURE CODE
747FH	1BH	TYPE OF FUNCTION

SYMBOL TABLE FUNCTION ENTRY

FIGURE 22.

```
                     |    Symbol    |
                     |    Table     |
                     |--------------|
                     |     4CH      |    \
                     |--------------|     \
                     |     4FH      |      \
                     |--------------|       \     PROCEDURE "LC"
                     |     02H      |        >
                     |--------------|       /     SYMBOL TABLE ENTRY
POINTER TO      /    |     97H      |      /
PARAMETER   :--<     |--------------|     /
LISTING     |   \    |     74H      |    /
            |        |--------------|
            |
            |        |--------------|
            |        |     01H      |    \   VALUE PARAMETER "X"
            |        |--------------|     >
            |        |     58H      |    /   SYMBOL TABLE ENTRY
            |        |--------------|
            |
            |        |--------------|
            |        |     01H      |    \   VALUE PARAMETER "Y"
            |        |--------------|     >
            |        |     59H      |    /   SYMBOL TABLE ENTRY
            |        |--------------|
            |
            |        |--------------|
            |---->   |     03H      |
                     |--------------|
                     |     24H      |    \   PRT LOCATION
                     |--------------|     >
                     |     00H      |    /   ASSIGNED
                     |--------------|
                     |     83H      |
                     |--------------|
                     |     26H      |    \   PRT LOCATION
                     |--------------|     >
                     |     00H      |    /   ASSIGNED
                     |--------------|
                     |              |
                     |--------------|
                     |              |
```

PROCEDURE AND PARAMETERS SYMBOL TABLE ENTRY

FIGURE 23.

scanner, analyzing them to determine if they are a part of the NPS-PASCAL grammar, then accepts or rejects the token according to the grammar. If the token is accepted, one of two actions is taken. The parser may stack the token and continue to request tokens in the lookahead state, or it may recognize the right part of a valid production and apply the production state. This results in a stack reduction. If the parser rejects the token, or determines that the token received does not constitute a valid right part of any production in the grammar, a syntax error message will be printed to the console and the RECOVER procedure is called.

RECOVER is a procedure that permits continued program compilation in spite of the detection of a syntax error. The parser backs up one state and attempts to continue parsing from that state. In the event of failure, the parser continues to back up until the end of the currently pending production is located. At that point the invalid token is completely bypassed, and an attempt is made to parse the following token. This process continues until an acceptable token is found.

The parse stacks in NPS-PASCAL consist of a state stack and eight auxiliary stacks. The auxiliary stacks are parallel to the parse stack and are used to store information extracted from the symbol table needed during code generation. The stacks are:

BASE$LOC - stores the symbol table address of the current identifier;

FORM$FIELD - store the form field value of the current identifier as reflected in the symbol table;

TYPE$STACK - stores the type value of the identifier;

PRT$ADDR - stores the PRT address of the identifier;

LABEL$STACK - stores the label value to be used with branching instructions;

PARM$NUM - stores the number of formal parameters associated with a procedure or function;

PARMNUMLOC - stores the sybol table address of the list of formal parameter types associated with a procedure or function;

EXPRESS$STK - stores the type value of an expression.

F. CODE GENERATION

The parser not only verifies the syntax of the source statements, but also controls the generation of the intermediate code by associating semantic actions with production rules. When a reduction takes place, the SYNTHESIZE procedure (in SYNTH2.SRC) is called with the production number as a parameter. The SYNTHESIZE procedure contains an extensive case statement keyed by the production number to perform the appropriate semantic actions. The syntax of the language and the semantic actions for each reduction are contained within the listing of the module SYNTH2.SRC.

Fundamental to understanding the compiler is a detailed knowledge of the NPS-PASCAL data structures, the pseudo operators, the use of procedures and functions, and the communication paths between the compiler and the user. The pseudo operators are described in detail in Ref. 2. These other elements are described below to assist in understanding the NPS-PASCAL compiler constructs and to explain the logic used to generate the intermediate code. That code will later be used to generate the target machine code.

1. Storage Space Allocation

The amount of storage allocated to a variable is a function of the type of the item. For each program variable requiring storage space, the compiler specifies the number of bytes to be alloted, and keeps a running total of the number of bytes assigned. The total count is then passed to the code generator to establish the size of the Program Reference Table (PRT).

a. Byte Data

Byte data items are stored in a single byte in memory. Byte data items can represent characters, numbers, or boolean variables.

b. Integer Data

Integers are represented by two byte locations in memory with the high order byte preceding the low order byte of the integer number. The storage design imitates the function of the 8080A microprocessor [8] in its movement of

data from memory or from the stack into the processors double byte registers during program execution. Integers are represented in two's complement form, with the high order bit acting as the sign bit. A zero high order bit indicates a positive integer, while a high order bit of one indicate a negative number.

c. Real Data

Real numbers are represented in binary coded decimal (BCD) format. Each real number is represented by fourteen decimal digits and is stored in eight consecutive bytes. When loading a BCD value onto the execution stack, the byte located at the lowest memory address contains the sign of the number along with the sign and magnitude of the exponent. Succeding bytes represent two decimal digits and are ordered backwards, such that the byte closest to the exponent byte contains the last two decimal digits of the number, while the last byte contains the left-most two decimal digits of the number. The format of a BCD number in memory is displayed in Fig. 24.

The exponent byte in a BCD number uses the high order bit to indicate the sign of the number: a one indicates positive, a zero negative. The remaining seven bits represent the exponent and its sign. The exponent is biased by 64 so that values greater than 64 (in seven bits) depict a positive exponent and values less than 64 depict a negative exponent; the exponent is the difference between 64 and the actual value. The bias allows exponent values

REPRESENTATION OF 12.3456789

1.23456789 X 10**1

.123456789E02

Memory Address		
1101H		
1102H	0	42H
1103H	0H	0H
1104H	0H	0H
1105H	9H	0H
1106H	7H	8H
1107H	5H	6H
1108H	3H	4H
1109H	1H	2H
100AH		

BCD NUMBER IN MEMORY

FIGURE 24.

ranging from -64 to +63. The BCD number always assumes that the decimal point is normalized before the first digit.

d. String Data

Strings are stored sequentially. The first byte of the string stores the string length, thus limiting strings to a length of 255 bytes. Immediately following the length byte are the ASCII characters of the string.

2. <u>Arithmetic Operations</u>

a. Logical Operations

Logical, or boolean, operations act on byte values of zero and one only. A zero value indicates a false condition, while a non-zero value indicates true. Logical operations requiring comparison between two elements returns the value of the operation in the true or false form.

b. Integers

Arithmetic operations with integers are performed by taking the top two values from the execution stack, and placing them in the double byte registers in the 8080 microprocessor, and then carrying out the requested operation using the microprocessors native functions. Integer arithmetic includes addition, subtraction, multiplication, division with truncation, modulo division, logical comparisons, and transformations to real (BCD) format. All computation results, except for real transformations, are returned to the execution stack in the two byte integer format. Relational operations on two

integer values are carried out in accordance with the rules for integer arithmetic.

c. Reals

Real arithmetic operations are more complex than those with integers due to the nature of the BCD format. The process is similar to that of integers in that pairs of real number bytes are moved to the 8080 registers. The required operation is performed, and the resulting real value is returned to the execution stack in the eight byte BCD format. Real values also follow the rules of integer arithmetic when involved in relational operations.

3. Set Operations

The set operations of set union, set difference, set intersection, set equality and inequality, set inclusion and set membership are not implemented in this version of NPS-PASCAL.

4. String Operations

The relational operators of equality and inequality have been implemented for strings. The remainder of the relational operators denote lexicographic ordering according to the character set ordering, and are not implemented in this version of NPS-PASCAL.

5. Procedures and functions

Procedures and functions, also called subroutines, give NPS-PASCAL the ability to display program segments as explicit subprograms. The only difference between a procedure and a function is that the function returns a

value to the top of the execution stack after it is invoked; a procedure does not. This means that a function call actually represents an arithmetic expression. Procedure calls, however, stand alone as program statements. An analysis of the following procedure and function implementation by Blanton and Moore [9] concluded that the current design is inadequate. Insufficient information is passed to allow parameter mapping from the execution stack to the PRT.

a. Invocation

Procedures and functions can be invoked with zero or more actual parameters. The list of actual parameters is substituted into the corresponding list of formal parameters declared in the procedure or function definition. If the formal parameter is a variable parameter, the actual parameter has to be a variable also. Should the formal parameter be a value parameter, then the actual parameter can be an expression, provided that the expression type matches the formal parameter type. For procedure and function formal parameters, the actual parameter must be a procedure or function identifier. Actual parameter types are checked against formal parameter types stored in the symbol table during program compilation. The method of passing actual parameters' values is via the execution stack. The procedure or function's memory location is generated in the form PRO <label>, where PRO is a mnemonic meaning "branch to

subroutine", and <label> is the label value stored in the subroutine's symbol table entry.

b. Storage Allocation

All parameters and variables declared within a procedure or function are assigned a location in the PRT. These locations immediately follow the PRT location of the procedure or function identifier. Upon recognition of a complete subroutine, another PRT location is allocated. This location is called the Save Block Pointer (SBP) for the subroutine. The PRT locations extending from the subroutines's identifier location through the SBPr make up a Pocedure Control Block (PCB). The effect is that the PCB is a contiguous set of PRT cells, as seen in Fig 25. The PCB construct is based on the one used in ALGOL-E [10], and its usefulness is in recursive calls to a procedure or function.

c. Parameter Mapping

NPS-PASCAL uses a scheme similar to ALGOL-E [10] in mapping the actual parameters of a procedure or function into its formal parameters. After recognition of a subroutine identifier, the actual parameters that are identifiers have their intermediate code generated in the form of a "PARM" or "PARMV" mnemonic followed by the PRT location of the actual parameter. These mnemonics load the execution stack with the values of the actual parameters. If the actual parameter is an expression, the expression result will be loaded automatically on top of the execution stack. Consequently, the compiler generates the mnemonic "PARMX"

```
E: CHAR;

ION ZERO (F, A: REAL): REAL;
R X, Z: REAL;
  SX: BOOLEAN;


      |           |
      |-----------|
      |           |
      |-----------|
P     |     Ø     |                          \
      |-----------|                           \
      |           |    \                       \
      |-----------|     \  LOCAL                \   PROCEDURE
      |           |      >                       \
      |-----------|     /  VARIABLES              \
      |           |    /                           >   CONTROL
      |-----------|                               /
      |           |    \  FORMAL                 /
      |-----------|     >                       /   BLOCK
      |           |    /  PARAMETERS           /
      |-----------|                           /
RO    |           |    FUNCTION NAME         /
      |-----------|
      |           |
      |-----------|
      |           |
      |-----------|
      |           |
```

A PROCEDURE CONTROL BLOCK

FIGURE 25.

after recognizing a complete expression that is acting as a value parameter. PARMX will not require any action by the code generator.

With the actual parameter in place, program control will branch to the procedure or function itself. The compiler generates code to place three items on top of the execution stack. The first item is the number of formal parameters (f) in the subroutine, the second is the PRT location of the subroutine's identifier (IDLOC), and the third is the SBP address in the PRT (SBPLOC) of the subroutine. The compiler then generates the SAVP operator, followed by the total byte count of PRT storage (t) assigned for the subroutine's identifier and all formal parameters. This is followed by a listing of byte storage required by each formal parameter (Pi) in the PRT in descending order. The execution of the SAVP operator is expected to cause the following actions to be generated by the code generator.

(1) The SBP location is examined

(a) if SBP = Ø then SBP := 1, else

(b) SBP > Ø and segment length (SBPLOC - IDLOC + 2) is obtained from the top of available memory, for example, at address x. The PCB is then copied from the PRT to the memory segment at x. The contents of the segment at x is then called the Save Block (SB). SBP := x.

(2) The top two elements of the execution stack are deleted; the next element (f) is copied and deleted from the stack; Pi = p(1).

(3) If f = Ø then halt. All actual parameters have been copied into the formal parameter locations in the PCB.

(4) PRT location (IDLOC + t - p(i)) := top of execution stack; delete the top element of the execution stack; t := t - p(i); p(i) := p(i) + 1.

(5) f := f - 1; go to step (3).

This process ensures that recursively calling a subroutine will not destroy the local variables and parameters of any preceding calls.

d. Function Return Value

Coupled with the SAVP operator is the UNSP (unsave) operator that reverses the actions of SAVP. Two parameters are required at the top of the stack, the SBP locations in the PRT (SBPLOC), and the PRT location of the subroutine identifier (IDLOC). The actions, then, of UNSP are.

(1) The value stored at IDLOC is copied to the top of the stack (this returns a value for the function calls; this value will be deleted for procedure calls).

(2) If the value of SBPLOC is greater than 1 then the SB at location SBPLOC in the free memory area is copied back to the PCB and the memory is

```
VAR Y: INTEGER;
 .
 .
 .
PROCEDURE LO (X: INTEGER; VAR Y: INTEGER);
    VAR TEMP: REAL;
    BEGIN
        TEMP:=SQRT(X)
        Y:=TRUNC(TEMP);
    END;
D:=6;
 .
 .
 .
LO(49,D);
 .
 .
 .
```

```
      STACK                                  PRT
      -----                                 -----
                  BEFORE SAVP
     |    |                          SBP |  0   |
rs-  | 28 | SBP in PRT              TEMP |  -   |
     | 22 | LO in PRT                  Y |  -   |
     |  2 | # Parameters               X |  -   |
     |  6 |   Actual                  LO |  -   |
     | 49 |   Paraeters                D |  6   |

            AFTER SAVP, BEFORE UNSP

     |    |                          SBP |  1   |
     |    |                         TEMP |  -   |
     |    |                            Y |  6   |
     |    |                            X | 49   |
     |    |                           LO |  -   |
rs-  |    |                            D |  6   |

                  AFTER UNSP

     |    |                          SBP |  0   |
     |    |                         TEMP | 7.0  |
     |    |                            Y |  7   |
     |    |                            X | 49   |
     |    |                           LO |  -   |
rs-  |    |                            D |  7   |
```

FIGURE 26.

freed. If SBP = 1 then SBP := 0. Consequently, the UNSP operator returns a value from function calls, and restores the PCB in the event of recursive calls. Figure 26 shows the actions of the SAVP and UNSP operators on the PRT and the execution stack.

e. Forward Declared Procedures and Functions

To permit the invocation of a procedure or function prior to its definition NPS-PASCAL utilizes a forward reference. The forward reference consists of the procedure (function) head, followed by the word FORWARD. When the procedure (function) is defined later in the program, the parameters are not repeated. FORWARD is not a reserved word in NPS-PASCAL. It is instead referred to as a directive. Directives are identifiers in NPS-PASCAL, that can only occur immediately after a procedure or function heading. Directives are contained in the BUILTINTBL.

f. Standard Procedures and Functions

The built-in procedures and functions that currently exist in NPS-PASCAL correspond to the standard procedures and functions specified in STANDARD PASCAL. Their operation, however, is considerably different from user defined procedures and functions. The compiler first generates code for any subroutine actual parameters. A mnemonic for the built-in procedure or function is then generated which tells the interpreter/translator that it must remove the parameters fom the execution stack, perform

the requested operation, and return the result to the stack. The standard procedures for input and output (Read, Readln, Write, and Writeln) will not require special action to be taken by the interpreter/translator. The remaining standard procedures dealing with files and pointer variables generate mnemonics that will require action by the interpreter/translator.

6. Input-Output

Input and output (I/O) can be handled in two ways: via console and via disk. Console I/O refers to the device the NPS-PASCAL user is utilizing to provide commands to the system -- usually a CRT terminal or teletype. Disk I/O refers to utilizing auxiliary files on the disk for data manipulation.

Input from consle I/O is achieved through READ or READLN statements. Console output is accomplished by the WRITE and WRITELN statements. Input to the console is accomplished by an operating system routine that reads one full console line into an input buffer. The code generator generates code to examine the buffer and convert ASCII characters contained within the buffer into appropriate NPS-PASCAL internal integer, real or string format. The input value is associated with the appropriate read statement variable parameter and then stored in the memory location allocated for that variable. A write statement takes the internal representations of integer, decimal, or byte values and converts them to their ASCII character

format. These values are then passed to an operating system print routine for console output. Constants and string variables are stored as ASCII strings in the intermediate code and the interpreter/translator will generate code to send them character by character to the system print routine.

Disk I/O is achieved through the same read and write statements utilized for console I/O. However, to read data from a disk file requires that the file identifier be specified as the first parameter in a read statement's list of actual parameters. The file identifier has to be specified in the same manner for disk write statements as well. The file identifiers used in read and write statements must be declared in a variable declaration part of a program block, or as a program parameter in the program declaration (called an external file). The file identifier has a specific PRT entry assigned by the compiler. At program execution, space will have to be allocated on the NPS-PASCAL stack for the File Control Block (FCB) information necessary to interface file operations with the operating system. Additionally, space should be provided for a 128 byte I/O buffer for every declared file.

7. NPS-PASCAL Pseudo Operators

A complete description of each of the NPS-PASCAL pseudo operators is presented in Ref. 2.

III. PROBLEMS IDENTIFIED AND CORRECTED

As noted in Ref. 2, the BUILTINTBL must be located at memory location 0106H in the executable module, since the collision field and previous entry addresses are calculated and entered by hand. Care must be taken during the LINK and LOCATE programs to ensure that the BUILTINTBL is located properly. Since the LINK program adds object modules together linearly, it is necessary to specify TABLES.OBJ as the first module in the command line to the LINK program. While organizing the LINKed together modules and adjusting the address into absolute code, the LOCATE program uses a default order of CODE, STACK, DATA, MEMORY. Constants in the PLM-80 source program (distinguished from variables by the DATA directive), however, are allocated memory first, before any executable code. Forcing the memory address assignments to start at 0103H with the directive (CODE 103H) to the LOCATE program places BUILTINTBL at 103H, so a three byte dummy field was added right before the BUILTINTBL declaration. The first three bytes of the final CP/M executable file (100H, 101H and 102H) are used to store a jump instruction which points to the compiler entry point.

The two previous theses used an 8080 simulator which ran on the IBM 360 and zeroed memory prior to loading the compiler. Many of the variables were not initialized, instead, relying on a zeroed memory location for their value. PL/M-80 includes two directives, INITIAL and DATA,

which are used to set the initial value of variables and constants, respectively.

An additional difference between PL/M and PL/M-80 is that the latter allows an implicit dimension specifier. This allows the table declarations in TABLES.SRC and other long declarations to be made without knowing or counting the exact length of the data string. The implicit dimension specifier is invoked by entering an asterisk instead of a decimal constant, i.e. (*) instead of (48).

Due to a deficiency in the grammar and its associated tables, a record structure was not recognized until the END statement was parsed. It was then too late to initialize the variables used to analyze each record declaration. As an interim fix, the code to handle a record declaration had been written into the scanner portion of the compiler. Contrary to the structure of the compiler, when a record declaration was recognized by the token number, the record initializing code was executed. Correcting this problem was the subject of a project undertaken by Anderson and Myers [10] during a course in compiler theory at the Naval Postgraduate School. As a result of their work, this code was removed from the scanner, and placed in the production case statement where it belongs. The grammar was corrected, the parse tables regenerated, and changes to the existing tables were made by comparing the listings and typing changes by hand. In the SYNTH2.SRC module, production 55 was changed from

<RECORD TYPE> ::= RECORD <FIELD LIST> END

to

<RECORD TYPE> ::= <RECORD> <FIELD LIST> END

and production 56 was added to read

<RECORD> ::= RECORD.

A record is now recognized when the token RECORD is parsed, and the initialization of variables takes place correctly. All the remaining productions were renumbered to properly reflect the parse tables.

The user assistance program SYMBOLTABLE provided by the last thesis effort failed in attempting to print the symbol table for nearly every test program tried. Considerable effort was expended during the current effort to debug, modify and upgrade this program to a useful tool. Code was added to determine the actual location in memory of the symbol table during the compilation, and the symbol table is moved to that address for processing. The SYMBOLTABLE program was eventually abandoned for a number of reasons. First, it was attempting to read sequentially entries in the symbol table which were designed to be accessed via the hash table. All too often, the program crashed because it was not able to locate the beginning of the next entry. More frequently, though, the entry in the symbol table was incorrect, causing the SYMBOLTABLE program to use incorrect pointers, lengths, codes, etc. The SYMBOLTABLE program was replaced by a much simpler, but much more useful program, called SYMDUMP, which is described in the next paragraph.

The CP/M utility DUMP was modified to print the contents of a file as a single column of hex character pairs, each representing a byte. Each pair is preceded by a four digit hex address, which corresponds to that byte's address in the symbol table. during compilation. The address of the beginning of the symbol table is a constant in the SYMDUMP program, and will have to be reset each time to reflect the new address of the symbol table whenever the compiler is changed. This necessitates reassembling SYMDUMP for each new version of the compiler, after determining the starting address of the symbol table from the previous SETBL entry address of the second entry. The output from the SYMDUMP program can be easily and efficiently scanned by hand to determine the contents of each entry. Collision address and previous entry adress pairs, for instance, can usually be recognized on sight. Since the program is not data-dependent, it cannot crash due to improper symbol table entries. A description of the changes to the CP/M utility DUMP.ASM is provided in Appendix C.

Examining the symbol tables from various test programs showed that the address of the parent type of simple variable declarations was not be entered properly. In production 86,

<IDENT VAR STRING> ::= <IDENTIFIER>,

code was added to save the parent type.

In the ENTRSUBNTRY procedure in SYNTH1.SRC, the procedure SUBR$CASE was being called twice for the same

limit (upper) of the subrange. Code was added to modify the second call to examine the lower limit and thus correctly determine the number of entries in the subrange.

In most case statements throughout the compiler, there is no range checking done on the variable used to index into a case statement. In PL/M-80, if the index evaluates to a number greater than the number of case statements available, the result is undefined. In other cases, semicolons representing no-operation cases were omitted, causing the wrong code to be executed for a given case. Code was added to direct the index to the correct case.

In a few instances, PL/M-80 address variables (16-bit) were being passed to byte variables(8-bit), resulting in the eight high-order bits being truncated and lost. The offending variable declarations were corrected.

When the compiler was broken into modules, there were a substantial number of variables declared PUBLIC and EXTERNAL needlessly. When a variable was used only in the module in which it was declared, the PUBLIC declaration was deleted. A number of subroutines were declared PUBLIC in one module and not called, and declared external and called from only one other module. These subroutines were moved to the calling module and not declared PUBLIC or EXTERNAL.

The displacement vector associated with each array dimension was being calculated incorrectly and it was stored in the same symbol table entry as the subrange. The array offset (for non-zero-origin array dimensions) was being

calculated incorrectly. Code was added to temporarily stack the array declarations and subsequently enter them into the symbol table correctly. Code was also added to calculate the array offset and the displacement vector for each dimension.

IV. REMAINING PROBLEMS

Signed identifier constant entries in the symbol table are identified as such by the FORM value 41H, but the sign is not stored or applied to the value of the constant.

Arrays were only examined for correct identification and entry into the symbol table. Arrays on the right side of the assignment statement are not handled properly.

Since no interpreter has been written, there is still no way to validate the intermediate code produced. The compiler will compile some small test programs without crashing, but it frequently will crash or go into infinite loops.

The code in the modules SYMBOL.SRC, SYNTH1.SRC and SYNTH2.SRC cannot be trusted to behave as described in the two previous thesis efforts. Each procedure needs to be examined on a line-by-line basis, with a possible eye toward rewriting substantial portions. In many, many cases, variables are ANDed or ORed with unexplained hex constants. The function of these constants should be determined and the hex constants should be named and documented. In many other instances, variables are shifted left or right and then again ANDed or ORed with hex constants. The shifting can be avoided by defining and documenting the appropriate masks. The global data base should be better organized, defined and documented. Variables enter procedures in unknown states, and are used or modified without range checking or any sort of validation. The ranges on case statement indices need to be checked before use, and each case should be a DO; END;

block, even if for a no-operation case, so that statements added will not introduce extra and erroneous cases.

V. CONCLUSIONS

NPS-PASCAL is still a long way from complete implementation. Major problems exist in the parse stack structure, in semantic action subroutines and in the symbol table construction and access. The groundwork for a viable PASCAL compiler has been started, but the compiler design needs a critical review and analysis.

The operation of this compiler is still dependent on the development of an 8080 interpreter or translator to validate the pseudo operators generated. Completing the NPS-PASCAL compiler project will require a substantial investment of study and time.

APPENDIX A - Compiler Error Messages

AD	Array dimension stack overflow: Simplify array declaration.
AN	Array nest overflow: Simplify declaration.
AT	Assignment type error: Type of expression not compatible with assignment variable type.
CE	Invalid expression: The variable types within the expression are not compatible.
CV	Incorrect control variable: The control variable has not been declared or is of type REAL.
DC	Duplicate constant name: Constant identifiers must be unique.
DE	Disk error: Recompile.
DT	Duplicate type name: Type identifiers must be unique.
EE	Exponent size error:
ET	Invalid expression type: The types of the variables used in an expression are incompatible.
IA	Invalid array index: Array index types must be scalar - INTEGER or REAL types are invalid.
IC	Invalid constant variable: Constant entry in symbol table is invalid.
IE	Integer size error:
IP	Improper parameter: The actual parameter type does not match the formal parameter type.
IR	Invalid read variable: Only INTEGER, REAL or STRING values can be read.
IS	Invalid subrange error: Check type and limits of declared subrange.
IT	Invalid type error: Array component type specification invalid.
IV	Variant stack overflow: Reduce the number of variant cases.
LS	Label syntax error: All labels must be integers.

NC Incorrect character:

NE Incorrect actual parameter: The actual parameter must be a variable and not an expression.

NP No production: Syntax error in source line.

NS Invalid set element: Set elements must be scalar.

PE Parameter error: This parameter format can only be used in a write statement.

PN Incorrect number of parameters: The total number of actual parameters fails to equal the total number of formal parameters.

RN Record field stack overflow: Reduce the number of fields specified.

RT WRITE$STMT parameter error: The parameter has to be of type REAL.

SO State stack overflow: simplify program.

TI Invalid type identifier: Type identifier not previously declared.

TO Symbol table overflow: Reduce number of declarations.

UL Undefined label error: Label not declared in label statement.

UO Invalid unary operator: Variable type must be INTEGER, REAL or subrange of INTEGER.

UP Undeclared procedure: Procedure identifier not previously declared.

VN Variable declaration stack overflow: Reduce the number of variables declared per line.

VO Variable stack overflow: Reduce the length of variable printnames.

WP WRITE$STMT parameter error: The length parameter has to be of type integer.

APPENDIX B - Intermediate Code DECODE Program

The last thesis effort included a program called DECODE which will read the intermediate code file and convert the hex pseudo codes into the corresponding mnemonics. The parameters associated with certain operators, such as labels, branches and load immediate values are printed also. Integer and real numbers are converted to decimal format. Strings are displayed as ASCII characters.

To use the DECODE program, compile a PASCAL program ommitting the $C compiler toggle:

```
A>PASCAL TEST.PAS
```

When a successful compilation is complete, run the DECODE program on the intermediate file:

```
A>DECODE TEST.PIN
```

The contents of the intermediate file will be printed on the console.

APPENDIX C - SYMDUMP Symbol Table Display Program

A symbol table displaying program was developed to aid in examining the symbol table and debugging the compiler. It is based on the CP/M DUMP utility, and uses the starting address of the symbol table in memory.

To prepare the SYMDUMP program, the user must first use the standard CP/M utility DUMP to dump the symbol table file. In this dump, the user determines the starting address of the symbol table by examining the previous entry address of the second entry. This address will change whenever the compiler is altered, since the symbol table is assigned to the first available memory address after the compiler. Modify the CP/M utility DUMP as follows: after the label OPENOK, change the argument of the LXI H from 0 to the starting address of the symbol table; after the label GLOOP, delete the JNZ NONUM instruction. Rename, reassemble and reload the program. The SYMDUMP program is now ready to be used on the .SYM file produced by the compiler:

A>SYMDUMP PROGRAM.SYM

SYMDUMP produces a vertical listing of the symbol table, one byte per line; each byte is preceded by its address in the symbol table.

APPENDIX D - Compiler Source Code Structure

A. MODULARIZATION

The PL/M version of the NPS-PASCAL compiler contained over 4700 lines of source code. When the compiler was transferred to the Intel Microprocessor Development System (MDS) and the ISIS-II operating system, it was broken up into manageable modules according to function. Each module now has fewer than 1000 lines of code, so editing is facilitated, and corrections to the compiler can be implemented much more rapidly. The two largest modules take less than 15 minutes each to recompile. A recompiled module can then be linked with the remaining modules. Maintaining the compiler as a single, large file would have caused excessively long edit sessions, and a recompile time of over an hour.

There are seven modules, each in a separate ISIS-II format file. SYSRTS.SRC contains the interface to the CP/M operating system, including the disk and console input-output procedures, and the GETCHAR procedure. SCAN.SRC contains the input scanner. PARSER.SRC contains the parser and its supporting procedures, and most of the global variables. TABLES.SRC contains the built-in symbol table and the parse tables. SYMBOL.SRC consists of procedures which manipulate the symbol table, either writing into or reading from individual entries. SYNTH1.SRC contains the code synthesizer, procedures which use the parse stacks and which generate the intermediate code. SYNTH2.SRC consists solely

of the production case statement. Source listings of the modules are provided following the appendices.

Modularizing the compiler introduced the PL/M-80 compiler directives PUBLIC and EXTERNAL. Any variable, function or procedure which is declared in one module, and referenced in another, must be declared PUBLIC in the first, and EXTERNAL in the second. Functions and procedures which have arguments must have those arguments in both declarations, also.

The XREF switch of the PL/M-80 compiler causes a cross-reference to be appended to the source listing. The cross-reference contains each source program identifier (literal, constant, variable, function or procedure) which occurs in the program, along with the line number of its defining occurrence, the line numbers of any references to it, and whether it is declared PUBLIC or EXTERNAL. This cross reference is a very useful tool for locating identifiers.

The IXREF switch of the PL/M-80 compiler causes a temporary file with an .IXI extension to be created, which contains information about each PUBLIC and EXTERNAL declaration in the source program. These .IXI files, one for each source module, are later collected and consolidated by the IXREF program, which produces an inter-module cross reference listing. This listing contains all PUBLIC and EXTERNAL identifiers, and names the module in which the identifier was declared PUBLIC, and lists all modules which

make an EXTERNAL reference to it. This list is also very useful during debugging.

B. LINKING AND LOCATING

The compiler, now separated into modules, must be recombined to form a body of executable code. This is accomplished by the LINK and LOCATE programs. The LINK program adds code from each of the modules and libraries referenced linearly, to form a single file. The LOCATE program locates the code at a particular address in memory and adjusts all of the relocatable addresses into absolute addresses.

C. TRANSFER FROM ISIS-II TO CP/M

Once the complete compiler has been located and adjusted, it needs to be transfered from the ISIS-II based system where the PL/M-80 compiler resides to a CP/M based system for execution. This is done with FROMISIS.COM, an undocumented program which runs under CP/M and reads a file from an ISIS-II format disk onto a CP/M format disk. The compiler is then processed by the undocumented program OBJCPM.COM, which strips off any symbol table information, adds a JMP instruction to the entry point to the beginning of the compiler, and creates the executable form of the compiler. The symbol table information is placed in separate files with .SYM and .LIN extensions. These files can be deleted if empty or not used, or they can be saved for use with the debugging tool SID.

D. EXECUTION AND DEBUGGING

When invoking NPS-PASCAL on a PASCAL program, the compiler is treated as any other program under CP/M. Along with the file name of the PASCAL program to be compiled, NPS-PASCAL accepts up to four switches which cause it to print to the console the PASCAL source code, the production numbers, the token numbers, and cause it to suppress creation of the intermediate file.

The facilities of SID, the CP/M Symbolic Instruction Debugger, permit run-time debugging and execution tracing of the compiler. To use SID, it is necessary to include the PL/M-80 compiler DEBUG switch when compiling the module of interest. The DEBUG switch causes the PLM-80 compiler to include identifier and line-number locations with the file. This information is later stripped out by the OBJCPM.COM program into the PASCAL.SYM and PASCAL.LIN files. These files are loaded by SID and used to reference and identify absolute machine addresses by symbolic expressions. Effective debugging of the compiler requires a detailed knowledge of the operation of SID as documented in the SID Users Manual. In transferring the compiler from PL/M to PL/M-80, it was necessary to shorten some of the identifier names to less than 16 characters to meet the requirements of SID.

In order to ascertain the proper operation of the compiler, it is also necessary to have accurate knowledge of the PASCAL language. To ensure testing the compiler with

programs of proper PASCAL syntax, most test programs were taken either from the Pascal Manual and Report or the Pascal Validation Suite.

The entire compilation, linking and loading, transfer to CP/M, execution and debugging process is documented by example in Appendix E.

APPENDIX E - COMPILE, LINK and LOCATE Instructions

This appendix provides step-by-step directions for compiling the NPS-PASCAL compiler, linking and locating the object modules, generating cross-reference listings, transfering the compiler to a CP/M based system, and executing and debugging the compiler. For additional information about the ISIS-II system, see Refs. 11 - 13. For additional information about operation under the CP/M system, see Ref. 14.

The NPS-PASCAL source files are compiled, linked and located under the ISIS-II operating system. First, compile each module with the appropriate switches to the PL/M-80 compiler:

```
-PLM80 :F1:SYSRTS.SRC XREF IXREF DATE(29 MAR 80) DEBUG
-PLM80 :F1:TABLES.SRC XREF IXREF DATE(29 MAR 80) DEBUG
-PLM80 :F1:PARSER.SRC XREF IXREF DATE(29 MAR 80) DEBUG
-PLM80 :F1:SCAN.SRC XREF IXREF DATE(29 MAR 80) DEBUG
-PLM80 :F1:SYMBOL.SRC XREF IXREF DATE(29 MAR 80) DEBUG
-PLM80 :F1:SYNTH1.SRC XREF IXREF DATE(29 MAR 80) DEBUG
-PLM80 :F1:SYNTH2.SRC XREF IXREF DATE(29 MAR 80) DEBUG
```

Due to space limitations on a single disk, it may be necessary to copy the .LST files to another disk as they are generated, or to redirect the .LST file to the :F0: disk with the PRINT switch:

```
-PLM80 :F1:SYNTH1.SRC XREF IXREF DEBUG PRINT(:F1:SYNTH1.LST)
```

Next, generate the inter-module cross-reference:

```
-IXREF :F1:*.IXI TITLE('NPS-PASCAL VER 0.0')
```

A printed copy of the inter-module cross reference is very useful during debugging.

A "SUBMIT" file has been created to facilitate the LINKing and LOCATing process. If a different LINKing or LOCATing command string is desired, it can, of course, be entered by hand. To invoke the prepared file:

```
-SUBMIT :F1:PASCAL
```

The file :F1:PASCAL.CSD used by the SUBMIT command contains the following command lines:

```
-DELETE :F1:PASCAL.LNK,:F1:PASCAL
-LINK :F1:TABLES.OBJ,:F1:SYSRTS.OBJ,:F1:SYMBOL.OBJ,&
:F1:SYNTH1.OBJ,:F1:SYNTH2.OBJ,:F1:PARSER.OBJ,:F1:PRINT.OBJ,&
:F1:SCAN.OBJ,:F0:PLM80.LIB TO :F1:PASCAL.LNK MAP
-LOCATE :F1:PASCAL.LNK CODE(1030H) MAP
```

Execution of these lines will create the files :F1:PASCAL.LNK and :F1:PASCAL.

Leaving the ISIS-II disk containing the NPS-PASCAL compiler in drive 1, insert and boot a CP/M disk in drive 0. The CP/M disk must contain, among other programs, the programs FROMISIS.COM, OBJCPM.COM and SID.COM. Transfer the NPS-PASCAL compiler from the ISIS-II disk to the CP/M disk:

```
A>FROMISIS PASCAL
```

Break out the .SYM and .LIN files and add the JMP instruction to locations 100H, 101H, and 102H:

```
A>OBJCPM PASCAL
```

This command will create three files from the PASCAL file: PASCAL.COM, the executable compiler, PASCAL.SYM and PASCAL.LIN, the files containing symbol table information for the run-time debugger SID.COM. When debugging with SID.COM, it is useful to have printed copies of the .SYM and .COM files. The file PASCAL (with no extension) can be deleted.

Create a PASCAL source program, for example TEST.PAS, with an available text editor. Invoke the NPS-PASCAL compiler:

```
A>PASCAL TEST.PAS
```

Up to four switches may be provided to the NPS-PASCAL compiler through the CP/M parms field immediately following the file specification:

```
A>PASCAL TEST.PAS $ABCD
```

The switches may appear in any order and have the following meanings:

A - List the source programs lines.
B - List the production numbers.
C - Suppress creation of the intermediate file.
D - List the token numbers.

To invoke the run-time debugger SID.COM:

```
A>SID PASCAL.COM PASCAL.SYM
SID VERS 1.4
SYMBOLS
NEXT  PC  END
6F00 0100 C278
#I* PASCAL.LIN
#R
#I TEST.PAS $ABCD
```

Then set up pass points, etc. and debug as necessary. For detailed instructions in the use of SID.COM, the run-time debugger, see Ref. 15.

APPENDIX F - Disk Directories

The NPS-PASCAL compiler is stored on two ISIS-II format disks with directories as follows.

The source files, the compiled object files, and the .IXI files are on the first disk:

```
DIRECTORY OF :F1:DEBUG
NAME   .EXT BLKS   LENGTH ATTR
COPY          65     8042
DIR           46     5733
TRINT .OBJ     2       70 W
SCAN   .SRC   83    10343
SYSRTS.IXI     6      549
SYSRTS.OBJ    43     5304
SYMBOL.SRC   241    30270
SCAN   .OBJ   31     3833
SYNTH2.SRC   400    50247
PARSER.OBJ    56     6968
CONVRT.SRC    37     4577
SYNTH1.SRC   445    55906
SYNTH1.OBJ   164    20570
SYNTH2.OBJ   135    16818
TABLES.OBJ    35     4347
PASCAL.CSD     3      227 W
PARSER.IXI    15     1742
SYSRTS.SRC    89    11123
SYNTH1.IXI    17     1980
PASCAL.LNK   480    60356
PASCAL       380    47692
SYNTH2.IXI    16     1837
SYMBOL.OBJ    87    10844
PARSER.SRC   112    13996
TABLES.SRC    79     9804
SCAN   .IXI    4      281
TABLES.IXI     3      136
SYMBOL.IXI    11     1264
```

The second disk consists solely of listing files, due to their large size:

```
DIRECTORY OF :F1:LISTIN
NAME  .EXT BLKS   LENGTH ATTR
SYSRTS.LST  245    30771
SCAN  .LST  197    24683
TABLES.LST  126    15794
PARSER.LST  306    38402
SYNTH1.LST 1023   128720
SYNTH2.LST  911   114569
SYMBOL.LST  543    68277
IMCR  .LST   99    12368
CONVRT.LST   68     8364
```

The ISIS-II system disk used during the development of the NPS-PASCAL compiler contains the following:

```
DIRECTORY OF :F0:ISIS
NAME  .EXT BLKS   LENGTH ATTR
TED         136    16951 W
TRINT .OBJ    2       70 W
COPY         65     8042 W
ASXREF       35     4239 W
ATTRIB       38     4682 W
BINOBJ       28     3399 W
DELETE       37     4506 W
DIR          46     5733 W
EDIT         56     6999 W
FORMAT       49     6093 W
HEXOBJ       35     4281 W
IDISK        50     6239 W
LIB          82    10227 W
LINK        114    14298 W
LOCATE      108    13505 W
OBJHEX       27     3284 W
RENAME       21     2487 W
SUBMIT       38     4629 W
PLM80       172    21605 W
TYPE          5      498 W
IXREF        82    10216 W
PLM80 .LIB   45     5615 W
PLM80 .OV0  150    18731 W
PLM80 .OV1  232    29122 W
PLM80 .OV2   66     8156 W
PLM80 .OV3  189    23706 W
PLM80 .OV4   72     8932 W
SYSTEM.LIB   24     2846 W
LINK  .OVL   29     3491 W
```

```
$PAGEWIDTH(80) TITLE('SYSRTS - SYSTEM SUPPORT ROUTINES')
 SYS$ROUTINES:DO;

/* CPM INTERFACE ROUTINES */

DECLARE LIT LITERALLY 'LITERALLY',
        EXT LIT 'EXTERNAL',
        CR LIT '13',
        LF LIT '0AH',
        DCL LIT 'DECLARE',
        PROC LIT 'PROCEDURE',
        TRUE LIT '1',
        ADDR LIT 'ADDRESS',
        FALSE LIT '0',
        FILEEOF LIT '1',
        FOREVER LIT 'WHILE TRUE';

DCL
        EOLCHAR LIT '0DH', /* END OF SOURCE LINE CHARACTER
*/
        TAB LIT '09H',
        SOURCERECSIZE LIT '128', /* SIZE OF SOURCE FILE
RECORD */
        INTRECSIZE LIT '128', /* INTERMEDIATE FILE RECORD
SIZE */
        CONBUFFSIZE LIT '82', /* SIZE OF CONSOLE BUFFER */
        EOFFILLER LIT '1AH'; /* CHAR FOR LAST RECORD ON FILE
*/
/***************************************************************/
/*** GLOBAL VARIABLES ***/
/***************************************************************/
DCL
              /* COMPILER TOGGLES */

         LIST$SOURCE BYTE EXT,
         NOINTFILE BYTE EXT,

   /* EXT VARIABLES */

        PRODUCTION BYTE EXT,
        TOKEN BYTE EXT,
        ACCUM(32) BYTE EXT,
        NEXTCHAR BYTE EXT,
        LAST$SBTBL$ID ADDR EXT,
            /* COUNTERS */
        EOFC LITERALLY '25',/* EOF */
        PARMS ADDR PUBLIC INITIAL(6DH),
        ERRORCOUNT ADDR PUBLIC INITIAL(0),
        CODESIZE ADDR PUBLIC INITIAL(0),
        DECI(4) ADDR INITIAL(1000,100,10,1);
   /*
```

```
*************************************************************
* *
* SYSTEM DEPENDENT ROUTINES AND VARIABLES *
* *
* THE FOLLOWING ROUTINES ARE USED BY THE COMPILER *
* TO ACCESS DISK FILES AND THE CONSOLE. THESE *
* ROUTINES ASSUME THE USE OF THE CP/M OPERATING *
* SYSTEM. *
* *
* THE FCB'S ARE USED BY THE SYSTEM TO MAINTAIN *
* INFORMATION ON OPEN FILES. THEY ARE ONLY USED BY *
* PROCEDURES IN THIS SECTION. THE BUFFERS AND POINTERS *
* TO THE BUFFERS ARE USED BY THE REMAINDER OF THE *
* PROGRAM, BUT THEIR SIZE MAYBE VARIED TO SUIT THE DISK *
* OPERATING SYSTEM BEING USED. *
* *
*************************************************************
  */ DCL
 /* NOTE: CP/M PROVIDES 5CH AS FCB AREA AND 80H AS A
BUFFER FOR
          PROGRAM USE */
      RFCBADDR ADDR INITIAL(5CH),
      RFCB BASED RFCBADDR(33) BYTE, /* SOURCE FCB */
      WFCB(33) BYTE /* INTERMEDIATE FILE FCB */
                    INITIAL (0,' ','PIN',0,0,0,0),
      SFCB(33) BYTE /* SYMBOL TABLE FILE FCB */
                    INITIAL (0,' ','SYM',0,0,0,0),
      SBLOC ADDR INITIAL(80H),
      SOURCEBUFF BASED SBLOC(SOURCERECSIZE) BYTE, /*
SOURCE BUFFER */
      SOURCEPTR BYTE INITIAL (SOURCERECSIZE), /* BUFFER
INDEX */
      DISKOUTBUFF(INTRECSIZE) BYTE,
      SYMOUTBUFF(INTRECSIZE) BYTE,
      BUFFPTR BYTE INITIAL(255), /* BUFFER INDEX */
      SYMBUFFPTR BYTE INITIAL(255), /* SBTBL BUFFER INDEX
*/
      LINEBUFF(CONBUFFSIZE) BYTE, /*CONSOLE OUT BUFFER */
      LINEPTR BYTE INITIAL(0), /* BUFFER INDEX */
      BDOS ADDR PUBLIC INITIAL(5H), /*JMP TO O/S ENTRY*/
      BOOT ADDR INITIAL(0H), /*REBOOT ENTRY*/
      LINENO ADDR, /* CURRENT LINE NUMBER */
      STARTBDOS ADDR PUBLIC INITIAL(6H);/*PTR TO START OF
BDOS*/


/***********************************************************/
/*** G L O B A L P R O C E D U R E S ***/
/***********************************************************/

MOVE: PROC (SOURCE,DESTIN,L) PUBLIC;
 /*MOVES FM SOURCE TO DESTIN FOR L BYTES */
  DCL (SOURCE,DESTIN) ADDR, /* L < 255 BYTES */
  (SCHAR BASED SOURCE, DCHAR BASED DESTIN,L) BYTE;
```

```
    DO WHILE (L:=L - 1) <> 255;
       DCHAR=SCHAR;
       DESTIN=DESTIN + 1;
       SOURCE=SOURCE + 1;
    END;
 END MOVE;

FILL: PROC (A,CHAR,N) PUBLIC; /* MOVE CHAR TO A N TIMES */
    DCL A ADDR,(CHAR,N,DEST BASED A) BYTE;
    DO WHILE (N := N -1) <> 255;
       DEST = CHAR;
       A = A + 1;
    END;
 END FILL;

/* MONITOR ROUTINES */

MON1:PROC(FUNC,INFO) EXT;
   DCL FUNC BYTE,
       INFO ADDR;
 END MON1;

MON2:PROC(FUNC,INFO) BYTE EXT;
   DCL FUNC BYTE,
       INFO ADDR;
 END MON2;

MON3:PROC PUBLIC;
   CALL BOOT;
 END MON3;

/* I/O ROUTINES */

PRINTCHAR:PROC(B) PUBLIC;
         /*S END THE ASCII CHARACTER B TO THE CONSOLE */
   DCL B BYTE;
   CALL MON1(2,B);
 END PRINTCHAR;

PRINT:PROC(A) PUBLIC;
     /* PRINT THE BUFFER STARTING AT ADDRESS A UNTIL '$' */
   DCL A ADDR;
   CALL MON1(9,A);
 END PRINT;

READ:PROC(A) PUBLIC;
    /* READ CONSOLE CHAR'S INTO BUFFER A */
   DCL A ADDR;
   CALL MON1(10,A);
 END READ;

CRLF:PROC PUBLIC;
    /* S END CARRIAGE-RETURN-LINE-FEED TO THE CONSOLE */
   CALL PRINTCHAR(CR);
```

```
   CALL PRINTCHAR(LF);
 END CRLF;

PRINTDEC: PROC(VALUE) PUBLIC;
    DCL VALUE ADDR, I BYTE, COUNT BYTE;
    DCL FLAG BYTE;
    FLAG = FALSE;
    DO I = Ø TO 3;
    COUNT = 3ØH;
       DO WHILE VALUE >= DECI(I);
          VALUE = VALUE - DECI(I);
          FLAG= TRUE;
          COUNT = COUNT + 1;
       END;
       IF FLAG OR (I>= 3) THEN
          CALL PRINTCHAR(COUNT);
       ELSE
          CALL PRINTCHAR(' ');
    END;
 END PRINTDEC;

PRINT$TOKEN:PROC PUBLIC;
   CALL PRINT(.(' TOKEN = $'));
   CALL PRINT$DEC(TOKEN);
   CALL PRINT(.(' $'));
 END PRINT$TOKEN;

PRINT$PROD: PROC PUBLIC;
   CALL PRINT(.(' PROD = $'));
   CALL PRINT$DEC(PRODUCTION);
   CALL PRINT(.(' $'));
 END PRINT$PROD;

PRINT$ERROR: PROC PUBLIC;
   CALL CRLF;
   CALL PRINTDEC(ERRORCOUNT);
   CALL PRINT(.(' ERROR(S) DETECTED',CR,LF,'$'));
 END PRINT$ERROR;

ERROR:PROC(ERRCODE) PUBLIC;
   DCL ERRCODE ADDR,
       I BYTE;

   ERRORCOUNT=ERRORCOUNT+1;
   CALL CRLF;
   CALL PRINT(.('*** $'));
   CALL PRINT$DEC(LINENO);
   CALL PRINT(.(' ERROR $'));
   CALL PRINTCHAR(HIGH(ERRCODE));
   CALL PRINTCHAR(LOW(ERRCODE));
   CALL PRINT(.(' NEAR $'));
   DO I=1 TO ACCUM(Ø);
     CALL PRINTCHAR(ACCUM(I));
   END;
```

```
   CALL PRINT(.(CR,LF,' AT ERROR $'));
   CALL PRINT$TOKEN;
   CALL PRINT$PROD;
   IF TOKEN=EOFC THEN
     DO;
       CALL PRINT$ERROR;
       CALL MON3;
     END;
 END ERROR;

DISKERR:PROC;
   CALL ERROR('DE');
   CALL MON3;
 END DISKERR;

OPEN$SRC$FILE: PROC PUBLIC;
    CALL MOVE(.('PAS'),RFCBADDR+9,3);
    RFCB(32),RFCB(12) = 0;
    IF MON2(15,RFCBADDR) = 255 THEN
       DO;
          CALL ERROR('NS');
          CALL MON3;
       END;
 END OPEN$SRC$FILE;

READ$SRC$FILE:PROC BYTE;
    DCL DCNT BYTE;
    IF (DCNT:=MON2(20,RFCBADDR)) > FILEEOF THEN
       CALL DISKERR;
    RETURN DCNT;
 END READ$SRC$FILE;

SETUP$INT$FIL:PROC PUBLIC;
   IF NOINTFILE THEN /*ONLY MAKE FILE IF TOGGLE OFF*/
     RETURN;
   CALL MOVE(.RFCB,.WFCB,9);
   CALL MON1(19,.WFCB);
   IF MON2(22,.WFCB)=255 THEN
     CALL DISKERR;
     /* SET UP SYMBOL TABLE FILE */
     CALL MOVE(.RFCB,.SFCB,9);
     SFCB(32)=0;
     WFCB(32) = 0;
     CALL MON1(19,.SFCB);
     IF MON2(22,.SFCB)=255 THEN
       CALL DISKERR;
 END SETUP$INT$FIL;

WRIT$INT$FILE:PROC PUBLIC;
   IF NOINTFILE THEN
     RETURN;
   CALL MON1(26,.DISKOUTBUFF);
   IF MON2(21,.WFCB)<>0 THEN
     CALL DISKERR;
```

```
   CALL MON1(26,80H); /* RESET DMA ADDR */
 END WRIT$INT$FILE;

EMIT: PROC(OBJCODE) PUBLIC;
   DCL OBJCODE BYTE;
   IF (BUFFPTR := BUFFPTR+1) >= INTRECSIZE THEN
     /*WRITE TO DISK*/
     DO;
       CALL WRIT$INT$FILE;
       BUFFPTR=0;
     END;
   DISKOUTBUFF(BUFFPTR)=OBJCODE;
 END EMIT;

GENERATE:PROC(OBJCODE) PUBLIC;
   DCL OBJCODE BYTE;
   CODESIZE=CODESIZE+1;
   CALL EMIT(OBJCODE);
 END GENERATE;

GEN$ADDR:PROC(A,B) PUBLIC;
   DCL A BYTE, B ADDR;
   CALL GENERATE(A);
   CALL GENERATE(LOW(B));
   CALL GENERATE(HIGH(B));
 END GEN$ADDR;

WRIT$SYM$FILE: PROC PUBLIC;
   IF NOINTFILE THEN
     RETURN;
   CALL MON1(26,.SYMOUTBUFF);
   IF MON2(21,.SFCB)<>0 THEN
     CALL DISKERR;
   CALL MON1(26,80H); /*RESET DMA ADDR*/
 END WRIT$SYM$FILE;

GEN$SYMTBL:PROC(OBJCODE) PUBLIC;
   DCL OBJCODE BYTE;
   IF (SYMBUFFPTR:=SYMBUFFPTR+1)>= INTRECSIZE THEN
     /*WRITE TO DISK*/
     DO;
       CALL WRIT$SYM$FILE;
       SYMBUFFPTR=0;
     END;
   SYMOUTBUFF(SYMBUFFPTR)=OBJCODE;
 END GEN$SYMTBL;

MOVE$SBTBL:PROC PUBLIC;
   DCL SYMPTR ADDR;
   DCL VALUE BASED SYMPTR BYTE;
   DO SYMPTR=.MEMORY TO (LAST$SBTBL$ID - 1);
       CALL GENSYMTBL(VALUE);
   END;
   CALL GENSYMTBL(0);
```

```
   CALL GENSYMTBL(0);
   CALL GENSYMTBL(0);
   CALL GENSYMTBL(0);
   CALL GENSYMTBL(EOFFILLER);
   CALL GENSYMTBL(EOFFILLER);
   CALL WRIT$SYM$FILE;
 END MOVE$SBTBL;


CLOSE$INT$FIL: PROC PUBLIC;
   /*CLOSE INT CODE FILE AND SYM TABLE FILE*/
   IF NOINTFILE THEN
     RETURN;
   IF MON2(16,.WFCB)=255 THEN
     CALL DISKERR;
   IF MON2(16,.SFCB)=255 THEN
     CALL DISKERR;
 END CLOSE$INT$FIL;


CLEAR$LN$BUFF:PROC PUBLIC;
    CALL FILL(.LINEBUFF,' ',CONBUFFSIZE);
 END CLEAR$LN$BUFF;

LISTLINE: PROC(LENGTH);
    DCL (LENGTH,I) BYTE;
    CALL CRLF;
    CALL PRINT$DEC(LINENO);
    CALL PRINT$CHAR(' ');
    DO I = 0 TO LENGTH;
       CALL PRINTCHAR(LINEBUFF(I));
    END;
    CALL CRLF;
 END LISTLINE;

/* SCANNER INTERFACE */

GETCHAR: PROC BYTE PUBLIC;
    NXT$SRC$CHAR: PROC BYTE;
        RETURN SOURCEBUFF(SOURCEPTR);
     END NXT$SRC$CHAR;

    CHECKFILE: PROC BYTE;
        DO FOREVER;
          IF (SOURCEPTR:=SOURCEPTR+1)>=SOURCERECSIZE THEN
              DO;
                 SOURCEPTR=0;
                 IF READ$SRC$FILE=FILEEOF THEN
                    RETURN TRUE;
              END;
           IF (NEXTCHAR:=NXT$SRC$CHAR)<>LF THEN
              RETURN FALSE;
        END;
     END CHECKFILE;
```

```
   IF CHECKFILE OR (NEXTCHAR = EOFFILLER) THEN
      DO; /* EOF REACHED */
         CALL MOVE(.('EOF',EOLCHAR,LF),SBLOC,5);
         SOURCEPTR = 0;
         NEXTCHAR=NXT$SRC$CHAR;
      END;
   LINEBUFF(LINEPTR:=LINEPTR + 1)=NEXTCHAR; /*OUTPUT LINE*/
   IF NEXTCHAR = EOLCHAR THEN
      DO;
         LINENO = LINENO + 1;
         IF LISTSOURCE THEN
            CALL LISTLINE(LINEPTR-1);
         LINEPTR = 0;
         CALL CLEARLINEBUFF;
      END;
   IF NEXTCHAR = TAB THEN
      NEXTCHAR = ' ';
   RETURN NEXTCHAR;
END GETCHAR;
END SYS$ROUTINES;
```

```
DECLARE LIT LITERALLY 'LITERALLY',
SCAN: DO;
        DCL LIT 'DECLARE',
        PROC LIT 'PROCEDURE',
        EXT LIT 'EXTERNAL',
        TRUE LIT '1',
        ADDR LIT 'ADDRESS',
        FALSE LIT '0',
        COMMENT LIT '7BH',
        UNCOMMENT LIT '7DH',
        FOREVER LIT 'WHILE TRUE';

DCL IDENTSIZE LIT '32', /* MAX IDENTIFIER SIZE + 1 */
        EOLCHAR LIT '0DH', /* END OF SOURCE LINE CHARACTER*/
        HASHMASK LIT '127', /* HASHTABLE SIZE -1 */
        STRINGDELIM LIT '27H', /*CHAR USED TO DELIMIT
STRINGS*/

   /*NUMBER TYPES */
        INTEGER$TYPE LIT '1',
        UNSIGN$EXPON LIT '3',
        REAL$TYPE LIT '2',
        SIGNED$EXPON LIT '4';
    /* GLOBAL VARIABLES */

DCL LIST$TOKEN BYTE PUBLIC INITIAL(FALSE),
        LIST$PROD BYTE PUBLIC INITIAL(FALSE),
        LIST$SOURCE BYTE PUBLIC INITIAL(FALSE),
        DEBUG$LN BYTE PUBLIC INITIAL(FALSE),
        NOINTFILE BYTE PUBLIC INITIAL(FALSE),

            /* GLOBAL VARIABLES USED BY THE SCANNER */

         TOKEN BYTE EXT, /* TYPE OF TOKEN JUST SCANNED */
         HASHCODE BYTE EXT, /* HASH VALUE OF CURRENT TOKEN
*/
         NEXTCHAR BYTE PUBLIC, /* CURRENT CHARACTER FROM
GETCHAR */
         CONT BYTE EXT, /* INDICATES FULL ACCUM--STILL MORE
*/
         ACCUM(IDENTSIZE) BYTE EXT, /* HOLDS CURRENT TOKEN
*/


        NUMBERC LIT '54',/* NUMBER */
        STRINGC LIT '55',/* STRING */
        IDENTC LIT '58';/* IDENTIFIER */

/* LOCAL VARIABLES */

DCL LOOKED BYTE,/*TRUE WHEN GETCHAR HAS ALREADY RETURNED A
```

```
CHAR*/
    TEMPCHAR1 BYTE, /* HOLDS PREVIOUSLY SCANNED CHAR */
    TEMPCHAR2 BYTE; DCL PARMLIST(9) BYTE INITIAL(' ');
DECLARE VOCAB(170) BYTE INITIAL
            (0, '.', '<', '(', '+', 5BH, '^', '*', ')', ';',
'-', '/', ',',
    '>', ':', '=', 5DH, '..', ':=', 'DO', 'IF', 'IN', 'OF',
'OR', 'TO', 'EOP',
    'AND', 'DIV', 'END', 'FOR', 'MOD', 'NIL', 'NOT', 'SET',
'VAR', 'CASE',
    'ELSE', 'FILE', 'GOTO', 'THEN', 'TYPE', 'WITH', 'ARRAY',
'BEGIN', 'CONST',
    'LABEL', 'UNTIL', 'WHILE', 'DOWNTO', 'PACKED', 'RECORD',
'REPEAT',
    'PROGRAM', 'FUNCTION', 'PROCEDURE'):

    DCL VLOC(10) BYTE
INITIAL(0,1,17,33,63,91,121,145,152,160);

    DCL VNUM(10) BYTE INITIAL(0,1,17,25,35,42,48,53,56,57);

    DCL COUNT(10) BYTE INITIAL(0,15,7,9,6,5,3,0,0,0);

/* GLOBAL PROCEDURES */

DECLARE PARMS ADDR EXTERNAL,
        TYPENUM BYTE EXTERNAL;

MOVE:PROC (SOURCE,DESTIN,L) EXTERNAL;
    DCL (SOURCE,DESTIN) ADDR,
             L BYTE;
 END MOVE;

ERROR:PROC(ERRCODE) EXTERNAL;
    DCL ERRCODE ADDR;
 END ERROR;

OPEN$SRC$FILE: PROC EXTERNAL;
 END OPEN$SRC$FILE;

CLEAR$LN$BUFF:PROC EXTERNAL;
 END CLEAR$LN$BUFF;

/**********************************************************************
 * SCANNER PROCEDURES *
 **********************************************************************/

GETCHAR: PROC BYTE EXTERNAL;
 END GETCHAR;

GETNOBLANK: PROC;
    DO WHILE((GETCHAR = ' ') OR (NEXTCHAR = EOLCHAR));
    END;
 END GETNOBLANK;
```

```
INIT$SCANNER: PROC PUBLIC;
    DCL COUNT BYTE,
        I BYTE;
    I=0;
    CALL MOVE(PARMS,.PARMLIST,8);
    IF PARMLIST(0)='$' THEN
      DO WHILE (COUNT:=PARMLIST(I:=I+1))<>' ';
         IF (COUNT:=COUNT-'A')<=4 THEN
           DO CASE COUNT;
              LISTSOURCE = TRUE; /* A */
              LISTPROD = TRUE; /* B */
              NOINTFILE = TRUE; /* C */
              LISTTOKEN = TRUE; /* D */
              DEBUGLN = TRUE; /* E */
           END; /* OF CASE */
      END;
    CALL OPEN$SRC$FILE;
    CALL CLEAR$LN$BUFF;
    CALL GETNOBLANK;
 END INIT$SCANNER;

/*****************************************************************
 * SCANNER *
 *****************************************************************/

SCANNER: PROC PUBLIC;

    PUTINACCUM: PROC;
        IF NOT CONT THEN
           DO;
              ACCUM(ACCUM(0) := ACCUM(0) + 1) = NEXTCHAR;
              HASHCODE = (HASHCODE+NEXTCHAR) AND HASHMASK;
              IF ACCUM(0) = 31 THEN CONT = TRUE;
           END;
     END PUTINACCUM;

    PUTANDGET: PROC;
        CALL PUTINACCUM;
        CALL GETNOBLANK;
     END PUTANDGET;

    PUTANDCHAR: PROC;
        CALL PUTINACCUM;
        NEXTCHAR = GETCHAR;
     END PUTANDCHAR;

    NUMERIC: PROC BYTE;
        RETURN(NEXTCHAR - '0') <= 9;
     END NUMERIC;

    LOWERCASE: PROC BYTE;
        RETURN (NEXTCHAR >= 61H) AND (NEXTCHAR <= 7AH);
     END LOWER$CASE;
```

```
DECIMALPT:PROC BYTE;
    RETURN NEXTCHAR='.';
 END DECIMALPT;

CONV$TO$UPPER:PROC;
    IF LOWERCASE THEN
       NEXTCHAR=NEXTCHAR AND 5FH;
 END CONV$TO$UPPER;

LETTER: PROC BYTE;
    CALL CONV$TO$UPPER;
    RETURN ((NEXTCHAR - 'A') <= 25);
 END LETTER;

ALPHANUM: PROC BYTE;
    RETURN NUMERIC OR LETTER ;
 END ALPHANUM;

SPOOLNUMRIC: PROC;
    DO WHILE NUMERIC;
       CALL PUTANDCHAR;
    END;
 END SPOOLNUMRIC;

SET$NEXT$CALL: PROC;
    IF (NEXTCHAR = ' ') OR (NEXTCHAR=EOLCHAR) THEN
       CALL GETNOBLANK;
    CONT = FALSE;
 END SET$NEXT$CALL;

LOOKUP: PROC BYTE;

 DCL MAXRWLNG LIT '9';
 DCL PTR ADDR, (FIELD BASED PTR) (9) BYTE;
 DCL I BYTE;

     COMPARE: PROC BYTE;
         DCL I BYTE;
         I = 0;
         DO WHILE (FIELD(I) = ACCUM(I := I + 1)) AND I
<= ACCUM(0);
         END;
         RETURN I > ACCUM(0);
      END COMPARE;

     IF ACCUM(0) > MAXRWLNG THEN
        RETURN FALSE;
     PTR=VLOC(ACCUM(0))+.VOCAB;
     DO I=VNUM(ACCUM(0)) TO
(VNUM(ACCUM(0))+COUNT(ACCUM(0)));
         IF COMPARE THEN
            DO;
               TOKEN=I;
```

```
                RETURN TRUE;
              END;
           PTR=PTR+ACCUM(Ø);
        END;
        RETURN FALSE;
     END LOOKUP;

CHECK$EXP: PROC;
        /* THIS TAKES CARE OF EXPON. FORM */
        IF NEXTCHAR = 'E' THEN
           DO;
              TYPENUM = UNSIGN$EXPON;
              CALL PUTANDCHAR;
              IF NEXTCHAR = '-' OR NEXTCHAR = '+' THEN
                 DO;
                    CALL PUTANDCHAR;
                    TYPENUM = SIGNED$EXPON;
                 END;
              CALL SPOOLNUMRIC;
           END;
 END CHECK$EXP;
/***************************************************************/ /***
SCANNER - MAIN CODE ***/
/***************************************************************/

    DO FOREVER;
      ACCUM(Ø), HASHCODE, TOKEN = Ø;
      IF (NEXTCHAR = STRINGDELIM) OR CONT THEN
      DO; /* FOUND STRING */
        TOKEN = STRINGC;
        CONT = FALSE;
        DO FOREVER;
          DO WHILE GETCHAR <> STRINGDELIM;
            CALL PUTINACCUM;
            IF CONT THEN RETURN;
          END;
          CALL GETNOBLANK;
          IF NEXTCHAR <> STRINGDELIM THEN
            RETURN;
          CALL PUT$IN$ACCUM;
        END; /* OF DO FOREVER */
      END; /* OF RECOGNIZING A STRING */
      ELSE IF NUMERIC THEN
      DO; /* HAVE DIGIT */
        TOKEN = NUMBERC;
        TYPENUM = INTEGER$TYPE;
        DO WHILE NEXTCHAR='Ø'; /*ELIM LEADING ZEROS*/
           NEXTCHAR=GETCHAR;
        END;
        CALL SPOOLNUMRIC;
        IF DECIMALPT THEN
           DO;
             TEMPCHAR1 = NEXTCHAR;
             NEXTCHAR = GETCHAR;
```

```
          IF DECIMALPT THEN
            DO;
              LOOKED=TRUE;/*HANDLE ..*/
              RETURN;
            END;
          ELSE
          DO;
               TEMPCHAR2 = NEXTCHAR;
               NEXTCHAR = TEMPCHAR1;
               CALL PUT$IN$ACCUM;
               NEXTCHAR=TEMPCHAR2;
               TYPENUM = REAL$TYPE;
               CALL SPOOLNUMPIC;
          END;
        END;
    CALL CHECK$EXP;
    IF ACCUM(0) = 0 THEN
      HASHCODE,ACCUM(ACCUM(0):=1) = '0';
    CALL SET$NEXT$CALL;
    RETURN;
 END; /* OF RECOGNIZING NUMERIC CONSTANT */
 ELSE IF LETTER THEN
DO; /* HAVE A LETTER */
   DO WHILE ALPHANUM;
      CALL PUTANDCHAR;
   END;
   IF NOT LOOKUP THEN
     TOKEN = IDENTC;
   CALL SET$NEXT$CALL;
   RETURN;
END; /* OF RECOGNIZING RW OR IDENT */
ELSE DO; /* SPECIAL CHARACTER */
 IF NEXTCHAR = COMMENT THEN
    DO;
       NEXTCHAR = GETCHAR;
       DO WHILE NEXTCHAR <> UNCOMMENT;
          NEXTCHAR = GETCHAR;
       END;
       CALL GET$NO$BLANK;
    END;
 ELSE
    DO;
       IF NEXTCHAR = ':' THEN
          DO;
             CALL PUTANDCHAR;
             IF NEXTCHAR = '=' THEN
                CALL PUTANDGET;
          END;
       ELSE
          IF NEXTCHAR = '.' THEN
             DO;
                IF LOOKED THEN
                  DO;
                     LOOKED=FALSE;
```

```
                        CALL PUT$IN$ACCUM;
                        NEXTCHAR='.';
                      END;
                    ELSE
                      CALL PUTANDCHAR;
                    IF NEXTCHAR = '.' THEN
                       CALL PUTANDGET;
                    ELSE
                      IF NUMERIC THEN
                       DO;
                          TOKEN = NUMBERC;
                          TYPENUM = REAL$TYPE;
                          CALL SPOOLNUMRIC ;
                          CALL CHECK$EXP;
                          CALL SET$NEXT$CALL ;
                          RETURN;
                       END;
                    END;
                 ELSE
                    CALL PUTANDGET;

              IF NOT LOOKUP THEN
                 CALL ERROR('NC');
              CALL SET$NEXT$CALL;
              RETURN;
           END;
       END; /* OF RECOGNIZING SPECIAL CHAR */
     END; /* OF DO FOREVER */
 END SCANNER;
END SCAN;
```

```
$PAGEWIDTH(80) TITLE(' PARSER')
 PARSER: DO;

DECLARE LIT LITERALLY 'LITERALLY',
        DCL LIT 'DECLARE', PUB LIT 'PUBLIC', EXT LIT
'EXTERNAL',
        PROC LIT 'PROCEDURE',
        TRUE LIT '1',
        ADDR LIT 'ADDRESS',
        FALSE LIT '0',
        FOREVER LIT 'WHILE TRUE',
        STATESIZE LIT 'ADDRESS',
        INDEXSIZE LIT 'ADDRESS'; DCL
        IDENTSIZE LIT '32', /* MAX IDENTIFIER SIZE - 1 */
        VARCSIZE LIT '100', /* SIZE OF VARC STACK*/
        PSTACKSIZE LIT '48', /* SIZE OF PARSE STACKS */
        HASHTBLSIZE LIT '128', /* SIZE OF HASHTABLE */
        BCDSIZE LIT '8', /* BYTES USED FOR BCD VALUES */
        MAX$NEST LIT '3', /*MAX LEVEL OF NESTS FOR TYPES*/
        MAX$ARRY$DIM LIT '5'; /* MAX ARRY DIMENSIONS */
        /* MANY OF THE FOLLOWING VARIABLES CAN BE REPLACED
BY
           MAKING USE OF THE PARALLEL PARSE STACKS */ DCL
        SIGNTYPE BYTE PUB INITIAL (0),
        CONST$TYPE BYTE PUB INITIAL (0),/* TYPE OF CONSTANT
*/
        FORM BYTE PUB INITIAL (0),
        EXPON BYTE PUB INITIAL (0),
        VECPTR BYTE PUB INITIAL (0),
        TYPENUM BYTE PUB INITIAL (0),
        CONST$PTR BYTE PUB INITIAL (0),
        TYPE$ADDR ADDR PUB INITIAL (0),
        TYPE$LOCT ADDR PUB INITIAL (0),
        VAR$PTR BYTE PUB INITIAL (0),
        VAR$PARM$PTR BYTE PUB INITIAL (0),
        ALOCBASICTYP BYTE PUB INITIAL (0),
        ARRY$QTY(MAX$ARRY$DIM) ADDR PUB INITIAL (0),
        VAR$BASE(10) ADDR PUB INITIAL (0),
        VAR$BASE1(10) ADDR PUB INITIAL (0),
        ALLC$QTY ADDR PUB INITIAL (0),
        TYPE$ORD$NUM BYTE PUB INITIAL (0),
        PARENT$TYPE ADDR PUB INITIAL (0),
        CONST$INDX BYTE PUB INITIAL (0),
        LOOKUP$ADDR ADDR PUB INITIAL (0),
        CONST$VEC(4) BYTE PUB INITIAL (0),
        CONST$VALUE(16) BYTE PUB INITIAL (0),
        CONST$PN$HASH(4) BYTE PUB INITIAL (0),
        CONST$PN$PTR BYTE PUB INITIAL (0),
        CONST$PN$SIZE(4) BYTE PUB INITIAL (0),
        INTEGER$DIFF ADDR PUB INITIAL (0),
        SUBR$VAL(2) ADDR PUB INITIAL (0),
```

```
        SUBR$TYPE(2) BYTE PUB INITIAL (0),
        SUBR$PTR BYTE PUB INITIAL (0),
        SUB$TYP$ADDR ADDR PUB INITIAL (0),
        SUBR$FORM BYTE PUB INITIAL (0),
        SIGNVALU BYTE PUB INITIAL (0),
        ARRY$BASE ADDR PUB INITIAL (0),
        ARRY$PTR BYTE PUB INITIAL (255), /* -1 */
        ARRY$DIM$PTR BYTE PUB INITIAL (0),
        PTRPTR BYTE PUB INITIAL (0),
        TAG$FD(MAX$NEST) BYTE PUB INITIAL (0),
        VAR$CAS$TP(MAX$NEST) ADDR PUB INITIAL (0),
        VAR$CAS$VAL(MAX$NEST) ADDR PUB INITIAL (0),
        REC$VAR$TYP(MAX$NEST) BYTE PUB INITIAL (0),
        REC$NST BYTE PUB INITIAL (255), /* -1 */
        RECORD$PTR BYTE PUB INITIAL (255), /* -1 */
        REC$ADDR(10) ADDR PUB INITIAL (0),
        REC$PAR$ADR(MAX$NEST) ADDR PUB INITIAL (0),
        VARIANT$PART(MAX$NEST) BYTE PUB INITIAL (0).
        FXD$OFST$BSE(MAX$NEST) ADDR PUB INITIAL (0),
        VAR$OFST$BSE(MAX$NEST) ADDR PUB INITIAL (0),
        CUR$OFST(MAX$NEST) ADDR PUB INITIAL (0),
        NUM$ARRY$DIM(MAX$ARRY$DIM) BYTE PUB INITIAL (0),
        ARRY$DIMEN(25) ADDR PUB INITIAL (0),
        ARY$DM$ADR$PTR BYTE PUB INITIAL (255). /* -1 */
        /* CASE STATEMENT VARIABLES */
        CASE$STK(12) BYTE PUB INITIAL (0),/* # OF STMTS IN
CURRENT CASE */
        CASE$COUNT BYTE PUB INITIAL (255), /* -1 - LEVEL OF
CASE STMTS */
        CONST$NUM$TYPE(4) BYTE PUB INITIAL (0); DCL
BCDNUM(BCDSIZE) BYTE PUB INITIAL (0),
        SCOPE(10) ADDR PUB INITIAL (0),
        SCOPE$NUM BYTE PUB INITIAL (0),
        TEMPBYTE BYTE PUB INITIAL (0),
        TEMPBYTE1 BYTE PUB INITIAL (0),
        TEMPADDR ADDR PUB INITIAL (0),
        TEMPADDR1 ADDR PUB INITIAL (0),
        PRODUCTION BYTE PUB INITIAL (0),
        PRV$SBT$ENTRY ADDR PUB INITIAL (0); DCL
              /* COMPILER TOGGLES */
 LIST$TOKEN BYTE EXT,
 COMPILING BYTE INITIAL (0),
 /* COUNTERS */
 LABLCOUNT ADDR PUB INITIAL (0), /* COUNTS NUMBER OF LABELS
*/
 ALLOC$ADDR ADDR PUB INITIAL (0), /* COUNTS PRT ENTRIES */
 /* FLAGS USED DURING CODE GENERATION */
 CASE$STMT BYTE PUB INITIAL (0), /* IN CASE STATEMENT */
 WRITE$STMT BYTE PUB INITIAL (0), /* IN WRITE STATEMENT */
 READ$STMT BYTE PUB INITIAL (0), /* IN READ STATEMENT */
 NEW$STMT BYTE PUB INITIAL (0), /* GETS NEW RECORD */
 DISPOSE$STMT BYTE PUB INITIAL (0), /* DISPOSES OF RECORD */
 ALLOCATE BYTE PUB INITIAL (0),/* PRT LOCATION ASSIGNED */
 VARPARM BYTE PUB INITIAL (0),/* FORMAL PARAM IS VARIABLE
```

```
TYPE */
 READPARMS BYTE PUB INITIAL (Ø),/* READING ACTUAL PARAMETERS
*/
 PRESENT BYTE PUB INITIAL (Ø),/* IDENTIFIER IS IN SYMBOL
TABLE */
 NO$LOOK BYTE INITIAL (Ø),/* CONTROLS CALLS TO SCANNER */
 SIGN$FLAG BYTE PUB INITIAL (Ø),/* SET WHEN SIGN PRECEDES ID
*/
 /* GLOBAL VARIABLES USED BY THE SCANNER */
 TOKEN BYTE PUB INITIAL (Ø), /* TYPE OF TOKEN JUST SCANNED
*/
 HASHCODE BYTE PUB INITIAL (Ø), /* HASH VALUE OF CURRENT
TOKEN */
 CONT BYTE PUB INITIAL (Ø), /* INDICATES FULL ACCUM--STILL
MORE */
 ACCUM(IDENTSIZE) BYTE PUB INITIAL (Ø), /* HOLDS CURRENT
TOKEN */
 /* GLOBAL VARIABLES USED IN SYMBOL TABLE OPERATIONS */
 BASE ADDR PUB INITIAL (Ø), /* BASE LOCATION OF ENTRY */
 HASHTABLE(HASHTBLSIZE) ADDR PUB INITIAL (Ø), /* HASHTABLE
ARRAY */
 SBTBLTOP ADDR PUB INITIAL (Ø), /* HIGHEST LOCATION OF
SYMBOL TABLE */
 SBTBL ADDR PUB INITIAL (Ø), /* CURRENT TOP OF SYMBOL TABLE
*/
 APTRADDR ADDR PUB INITIAL (Ø), /* UTILITY VARIABLE TO
ACCESS SBTBL */
 PRINTNAME ADDR PUB INITIAL (Ø), /* SET PRIOR TO LOOKUP OR
ENTER */
 SYMHASH BYTE PUB INITIAL (Ø), /* HASH VALUE OF AN
IDENTIFIER */
 LAST$SBTBL$ID ADDR PUB INITIAL (Ø), /* HOLD PREVIOUS BASE
LOCATION */
 PARAMNUMLOC ADDR PUB INITIAL (Ø), /* STORES POINTER TO
PARAM LISTING */
 SBTBLSCOPE ADDR PUB INITIAL (Ø), /* BASE OF LAST ENTRY IN
PREVIOUS BLOCK*/ BUILTINTBL(1Ø) BYTE EXT;

FILL: PROC (A,CHAR,N) EXT;
  DCL A ADDR,
      (CHAR,N) BYTE;
 END FILL;

INIT$SYMTBL: PROC; DCL SYMBASE ADDR;
       DO;
          CALL FILL(.HASHTABLE,Ø,255);
          SYMBASE=.BUILT$IN$TBL(Ø);
          SBTBL=.MEMORY;
          HASHTABLE(14)=SYMBASE;
          HASHTABLE(36)=SYMBASE+14;
          HASHTABLE(3Ø)=SYMBASE+25;
          HASHTABLE(Ø)=SYMBASE+36;
          HASHTABLE(69)=SYMBASE+5Ø;
          HASHTABLE(16)=SYMBASE+61;
```

```
          HASHTABLE(113)=SYMBASE+77;
          HASHTABLE(86)=SYMBASE+86;
          HASHTABLE(118)=SYMBASE+130;
          HASHTABLE(57)=SYMBASE+142;
          HASHTABLE(109)=SYMBASE+159;
          HASHTABLE(26)=SYMBASE+173;
          HASHTABLE(74)=SYMBASE+186;
          HASHTABLE(87)=SYMBASE+201;
          HASHTABLE(90)=SYMBASE+230;
          HASHTABLE(12)=SYMBASE+244;
          HASHTABLE(8)=SYMBASE+260;
          HASHTABLE(101)=SYMBASE+276;
          HASHTABLE(93)=SYMBASE+290;
          HASHTABLE(46)=SYMBASE+304;
          HASHTABLE(43)=SYMBASE+319;
          HASHTABLE(121)=SYMBASE+334;
          HASHTABLE(96)=SYMBASE+347;
          HASHTABLE(3)=SYMBASE+360;
          HASHTABLE(34)=SYMBASE+375;
          HASHTABLE(29)=SYMBASE+392;
          HASHTABLE(106)=SYMBASE+406;
          HASHTABLE(23)=SYMBASE+418;
          HASHTABLE(64)=SYMBASE+434;
          HASHTABLE(107)=SYMBASE+449;
          HASHTABLE(28)=SYMBASE+465;
          HASHTABLE(54)=SYMBASE+478;
          HASHTABLE(11)=SYMBASE+493;
          HASHTABLE(37)=SYMBASE+507;
          HASHTABLE(40)=SYMBASE+523;
          HASHTABLE(21)=SYMBASE+538;
          HASHTABLE(99)=SYMBASE+552;
          HASHTABLE(62)=SYMBASE+567;
          PRV$SBT$ENTRY = SYMBASE+567;
       END;
 END INIT$SYMTBL;


DCL STATE STATESIZE INITIAL (0),
        VAR(PSTACKSIZE) BYTE PUB INITIAL (0),
        HASH(PSTACKSIZE) BYTE PUB INITIAL (0),
        STATESTACK(PSTACKSIZE) STATESIZE INITIAL (0),
        PARMNUM(PSTACKSIZE) BYTE PUB INITIAL (0), /*
MAINTAINS NUMBER OF PARAMETERS
                                        ASSOCIATED WITH A
SUBROUTINE */
        LABELSTACK(PSTACKSIZE) ADDR PUB INITIAL (0), /*
TRACKS STATEMENT LABELS */
        PARMNUMLOC(PSTACKSIZE) ADDR PUB INITIAL (0), /*
MAINTAINS THE LOCATION IN SYMBOL
                                        TBL WHERE PARAMETER
INFO STORED */
        BASE$LOC(PSTACKSIZE) ADDR PUB INITIAL (0), /* STORES
THE SYMBOL TABLE ADDRESS
                                        OF THE PERTINATE ENTRY
```

```
*/
        FORM$FIELD(PSTACKSIZE) BYTE PUB INITIAL (0), /*
STORES THE FORM FIELD OF
                                        SCANNED IDENTIFIERS */
        TYPE$STACK(PSTACKSIZE)BYTE PUB INITIAL (0),/* HOLDS
A VARIABLE'S TYPE */
        EXPRESS$STK(PSTACKSIZE)BYTE PUB INITIAL (0), /*
CONTAINS THE TYPES OF THE
                                        EXPRESSION COMPONENTS
*/
        PRT$ADDR(PSTACKSIZE) ADDR PUB INITIAL (0), /* STORES
AN IDENTIFIER'S PRT
                                        LOCATION */
        VARC(VARCSIZE) BYTE PUB INITIAL (0),
        VARINDEX BYTE INITIAL (0),
        PARAMNUM BYTE PUB INITIAL (0),
        (SP,MP,MPP1) BYTE PUB INITIAL (0);
    /* MNEMONICS FOR PASCAL-SM MACHINE */
 DCL MAXRNO LIT '185' /*MAX READ COUNT*/, MAXLNO LIT '242'
/*MAX LOOK COUNT*/, MAXPNO LIT '269' /*MAX PUSH COUNT*/,
STARTS LIT '1' /*START STATE*/;
 DECLARE READ1(1) BYTE EXT, READ2(1) ADDR EXT, INDEX1(1)
ADDR EXT, INDEX2(1) BYTE EXT, APPLY1(1) BYTE EXT, APPLY2(1)
ADDR EXT, LOOK1(1) BYTE EXT, LOOK2(1) ADDR EXT;

SETUP$INT$FIL: PROC EXT; END SETUP$INT$FIL;

INIT$SCANNER: PROC EXT; END INIT$SCANNER;

INIT$SYNTH: PROC EXT; END INIT$SYNTH;

ERROR: PROC(ERRCODE) EXT; DECLARE ERRCODE ADDR; END ERROR;

SCANNER: PROC EXT; END SCANNER;

PRINT$TOKEN: PROC EXT; END PRINT$TOKEN;

SYNTHESIZE: PROC EXT; END SYNTHESIZE;

PRINT:PROC(A) EXT;
    DCL A ADDR;
 END PRINT;

CRLF:PROC EXT;
 END CRLF;

TITLE:PROC;
    CALL CRLF;
    CALL PRINT(.('NPS-PASCAL VERS 0.0 3-MAR-80 $'));
    CALL CRLF;
 END TITLE;

NOCONFLICT: PROC (CSTATE) BYTE;
    DCL CSTATE STATESIZE, (I,J,K) INDEXSIZE;
```

```
    J= INDEX1(CSTATE);
    K= J + INDEX2(CSTATE) - 1;
    DO I = J TO K;
       IF READ1(I) = TOKEN THEN RETURN TRUE;
    END;
    RETURN FALSE;
 END NOCONFLICT;

RECOVER: PROC STATESIZE;
    DCL TSP BYTE, RSTATE STATESIZE;
    DO FOREVER;
       TSP = SP;
       DO WHILE TSP <> 255;
          IF NOCONFLICT(RSTATE:=STATESTACK(TSP)) THEN
             DO; /* STATE WILL READ TOKEN */
                IF SP <> TSP THEN SP = TSP - 1;
                RETURN RSTATE;
             END;
          TSP = TSP - 1;
       END;
       CALL SCANNER;
    END;
 END RECOVER;

DO; /*BLOCK FOR DECLARATIONS*/
    DCL (I,J,K) INDEXSIZE, INDEX BYTE;
    INITIALIZE: PROC;
        CALL INIT$SCANNER;
        CALL INIT$SYMTBL;
          CALL INIT$SYNTH;
        CALL TITLE;
     END INITIALIZE;
    GETIN1: PROC INDEXSIZE;
        RETURN INDEX1(STATE);
     END GETIN1;
    GETIN2: PROC INDEXSIZE;
        RETURN INDEX2(STATE);
     END GETIN2;
    INCSP: PROC;
        IF (SP := SP + 1) = LENGTH(STATESTACK) THEN
              CALL ERROR('SO');
     END INCSP;
    LOOKAHEAD: PROC;
        IF NOLOOK THEN
           DO;
              CALL SCANNER;
              NOLOOK = FALSE;
              IF LISTTOKEN THEN
                 CALL PRINT$TOKEN;
           END;
     END LOOKAHEAD;

    SET$VARC$I: PROC(I); /* SET VARC, AND INCRMNT VARINDEX
*/
```

```
      DCL I BYTE;
      VARC(VARINDEX)=I;
      IF (VARINDEX:=VARINDEX+1) > LENGTH(VARC) THEN
         CALL ERROR('VO');
   END SET$VARC$I;


$EJECT /**************************************************/ /*
*/ /* PARSER: EXECUTION BEGINS HERE */ /* */
/**************************************************/


CALL SETUP$INT$FIL; /* CREATES OUTPUT FILE FOR GENERATED
CODE */ CALL INITIALIZE; COMPILING,NOLOCK=TRUE;
STATE=STARTS; SP=255; VARINDEX,VAR(0) = 0; DO WHILE
COMPILING;
  IF STATE<=MAXRNO THEN /* READ STATE */
    DO;
    CALL INCSP;
    STATESTACK(SP)=STATE;
    I=GETIN1;
    CALL LOOKAHEAD;
    J=I+GETIN2-1;
    DO I=I TO J;
      IF READ1(I)=TOKEN THEN /* SAVE TOKEN */
        DO; /* COPY ACCUM TO PROPER POSITION */
        VAR(SP)=VARINDEX;
        DO INDEX = 0 TO ACCUM(0);
           CALL SET$VARC$I(ACCUM(INDEX));
           END;
        HASH(SP) = HASHCODE;
        STATE=READ2(I);
        NOLOOK=TRUE;
        I=J;
        END;
      ELSE
        IF I=J THEN
          DO;
          CALL ERROR('NP');
          IF (STATE := RECOVER)=0 THEN
             COMPILING = FALSE;
          END;
      END;
    END;
  ELSE IF STATE>MAXPNO THEN /* APPLY PRODUCTION STATE */
    DO;
    MP=SP-GETIN2;
    MPP1=MP+1;
    PRODUCTION = STATE-MAXPNO;
    CALL SYNTHESIZE;
    SP=MP;
    I=GETIN1;
    VARINDEX=VAR(SP);
```

```
    J=STATESTACK(SP);
    DO WHILE (K:=APPLY1(I)) <> 0 AND J <> K;
      I=I+1;
      END;
    IF (STATE:= APPLY2(I))=0 THEN
      COMPILING = FALSE;
    END;
  ELSE
    IF STATE<= MAXLNO THEN /* LOOKAHEAD STATE */
      DO;
      I=GETIN1;
      CALL LOOKAHEAD;
      DO WHILE (K:=LOOK1(I)) <> 0 AND TOKEN <> K;
        I=I+1;
        END;
      STATE=LOOK2(I);
      END;
   ELSE
     DO; /* PUSH STATE */
     CALL INCSP;
     STATESTACK(SP)= GETIN2;
     STATE=GETIN1;
   END;
END; /* OF WHILE COMPILING */

END; /* OF BLOCK FOR PARSER */

END PARSER;
```

```
TABLES: DO;
 $PAGEWIDTH(80) TITLE('TABLES - LALR(1) PARSE TABLES')

DECLARE LIT LITERALLY 'LITERALLY',
        ADDR LIT 'ADDRESS',
        DCL LIT 'DECLARE',
        PUB LIT 'PUBLIC';

DCL DUMMY (3) BYTE DATA (Ø,Ø,Ø); /*DUMMY FILLER TO FORCE
BUILT$IN$TABLE TO 106H */

DCL BUILT$IN$TBL (*) BYTE PUB /*AT (106H)*/ DATA (
 Ø,Ø,Ø,Ø,42H,14,7,'I','N','T','E','G','E','R',
 Ø,Ø,Ø1H,Ø6H,4AH,36,4,'R','E','A','L',
 Ø,Ø,Ø1H,14H,52H,32,4,'C','H','A','R',
 Ø,Ø,Ø1H,1FH,5AH,Ø,7,'B','O','O','L','E','A','N',
 Ø,Ø,Ø1H,2AH,62H,69,4,'T','E','X','T',
 Ø,Ø,Ø1H,38H,ØEH,16,5,'I','N','P','U','T',
 Ø,Ø,Ø1H,43H,1EH,113,6,'O','U','T','P','U','T',
 Ø,Ø,Ø1H,4FH,ØDH,86,3,'A','B','S',Ø,13H,1,13H,
 Ø,Ø,Ø1H,5CH,ØDH,118,3,'S','Q','R',1,13H,1,13H,
 Ø,Ø,Ø1H,6AH,ØDH,106,3,'S','I','N',2,3H,1,13H,
 Ø,Ø,Ø1H,78H,ØDH,1Ø1,3,'C','O','S',3,3H,1,13H,
 Ø,Ø,Ø1H,86H,ØDH,57,6,'A','R','C','T','A','N',4,3H,1,13H,
 Ø,Ø,Ø1H,94H,ØDH,1Ø9,3,'E','X','P',5,3H,1,13H,
 Ø,Ø,Ø1H,ØA5H,ØDH,26,2,'L','N',6,3H,1,13H,
 Ø,Ø,Ø1H,ØB3H,ØDH,74,4,'S','Q','R','T',7,3H,1,13H,
 Ø,Ø,Ø1H,ØCØH,ØDH,87,3,'O','D','D',8,5H,1H,1H,
 Ø,Ø,Ø1H,ØCFH,ØDH,46,4,'E','O','L','N',9,5H,1,Ø6H,
 Ø,Ø,Ø1H,ØDDH,ØDH,9Ø,3,'E','O','F',1Ø,5H,1,Ø6H,
 Ø,Ø,Ø1H,ØECH,ØDH,12,5,'T','R','U','N','C',11,1H,1,3H,
 Ø,Ø,Ø1H,ØFAH,ØDH,8,5,'R','O','U','N','D',12,1H,1,3H,
 86H,Ø1H,Ø2H,ØAH,ØDH,1Ø1,3,'O','R','D',13,1H,1,2H,
 Ø,Ø,Ø2H,1AH,ØDH,93,3,'C','H','R',14,2H,1,1H,
 ØDDH,Ø1H,Ø2H,28H,ØDH,46,4,'S','U','C','C',15,ØF3H,1,ØF3H,
 Ø,Ø,Ø2H,36H,ØDH,43,4,'P','R','E','D',16,ØF3H,1,ØF3H,
 Ø,Ø,Ø2H,45H,ØCH,121,3,'P','U','T',17,1ØH,Ø6H,
 Ø,Ø,Ø2H,54H,ØCH,96,3,'G','E','T',18,1ØH,Ø6H,
 Ø,Ø,Ø2H,61H,ØCH,Ø3,5,'R','E','S','E','T',19,1ØH,Ø6H,
 Ø,Ø,Ø2H,6EH,ØCH,34,7,'R','E','W','R','I','T','E',2Ø,1ØH,
Ø6H,
 Ø,Ø,Ø2H,7DH,ØCH,29,4,'P','A','G','E',21,1ØH,Ø6H,
 78H,Ø1H,Ø2H,8EH,ØCH,1Ø6,3,'N','E','W',22,ØFFH,
 Ø,Ø,Ø2H,9CH,ØCH,23,7,'D','I','S','P','O','S','E',23,ØFFH,
 Ø,Ø,Ø2H,ØA8H,Ø9H,64,4,'T','R','U','E',Ø,Ø,Ø,Ø,
 Ø,Ø,Ø2H,ØB8H,Ø9H,1Ø7,5,'F','A','L','S','E',Ø,Ø,1,Ø,
 Ø,Ø,Ø2H,ØC7H,ØCH,28,4,'R','E','A','D',24,ØFFH,
 Ø,Ø,Ø2H,ØD7H,ØCH,54,6,'R','E','A','D','L','N',25,ØFFH,
 Ø,Ø,Ø2H,ØE4H,ØCH,11,5,'W','R','I','T','E',26,ØFFH,
 Ø,Ø,Ø2H,ØF3H,ØCH,37,7,'W','R','I','T','E','L','N',27,ØFFH,
 Ø,Ø,Ø3H,Ø1H,ØCH,4Ø,4,'S','E','E','K',28,2,Ø6H,Ø1H,
```

```
0,0,03H,11H,11H,21,7,'F','O','R','W','A','R','D',
0,0,03H,20H,11H,99,8,'E','X','T','E','R','N','A','L',
0,0,03H,2EH,11H,62,11,'I','N','T','E','R','A','C','T','I',
'V','E');


DCL READ1 (*) BYTE PUB DATA(0, 53, 56, 57, 25, 25, 25, 13,
15, 34, 56, 57, 58, 58, 58, 9, 14, 9, 58 , 58, 58, 58, 58,
15, 58, 4, 10, 54, 55, 58, 3, 4, 6, 10, 33, 37, 42, 49, 50,
54, 55, 58, 22 , 3, 4, 5, 10, 31, 32, 54, 55, 58, 3, 4, 10,
54, 55, 58, 58, 3, 5, 31, 32, 54, 55, 58, 22, 58 , 58, 58,
22, 58, 5, 20, 29, 35, 38, 41, 43, 47, 51, 54, 58, 58, 54,
33, 37, 42, 50, 58, 58 , 58, 58, 20, 29, 35, 38, 41, 43, 47,
51, 58, 40, 44, 34, 56, 57, 54, 58, 7, 11, 26 , 27, 30, 58,
1, 1, 1, 14, 43, 35, 58, 3, 9, 17, 3, 14, 15, 1, 5, 6, 18,
1, 3, 5, 6, 9, 36, 36, 39, 22 , 19, 8, 17, 14, 14, 28, 8, 9,
3, 9, 28, 9, 46, 22, 22, 3, 12, 16, 14, 15, 9, 9, 8, 12, 16,
9 , 12, 24, 48, 9, 9, 8, 12, 12, 9, 12, 14, 12, 14, 12, 3,
9, 8, 12, 8, 12, 18, 45, 58, 12, 14 , 12, 16, 12, 19, 2, 4,
10, 13, 15, 21, 23, 4, 10, 23, 9, 12, 14, 9, 28, 8, 9, 8, 9,
0, 0, 0, 0 );


DCL LOOK1 (*) BYTE PUB DATA(0, 13, 15, 0, 35, 58, 0, 16, 0,
58, 0, 58, 0, 58, 0, 35, 58, 0, 9, 28, 46 , 0, 9, 28, 0, 8,
9, 28, 0, 15, 0, 8, 9, 28, 0, 8, 9, 28, 0, 9, 28, 36, 46, 0,
36, 0, 9, 28, 0 , 17, 0, 1, 5, 6, 18, 0, 14, 0, 0, 0, 0, 40,
0, 44, 0, 34, 0, 43, 0, 7, 11, 26, 27 , 30, 0, 7, 11, 26,
27, 30, 0, 7, 11, 26, 27, 30, 0, 35, 58, 0, 9, 46, 0, 36, 0,
1, 3, 5, 6, 0, 12, 19, 0 , 12, 19, 0, 9, 28, 36, 46, 0, 36,
0, 9, 28, 46, 0, 17, 0, 14, 0, 14, 0, 9, 0, 9, 28, 0, 43, 0
, 9, 28, 0, 12, 0, 9, 0, 9, 0, 12, 0, 3, 0, 45, 0, 45, 0,
45, 58, 0, 45, 0, 45, 58, 0, 2, 4, 10 , 13, 15, 21, 23, 0,
4, 10, 23, 0);


DCL APPLY1 (*) BYTE PUB DATA(0, 0, 0, 0, 0, 32, 0, 170, 171,
174, 175, 0, 0, 0, 31, 74, 79, 0, 0, 0 , 23, 0, 0, 28, 29,
39, 51, 54, 61, 63, 150, 0, 95, 0, 15, 28, 29, 37, 39, 47,
48, 51, 52 , 54, 59, 60, 61, 62, 63, 150, 0, 0, 0, 24, 0, 0,
47, 48, 60, 62, 0, 15, 37, 59, 0, 18, 44 , 45, 46, 69, 123,
0, 0, 0, 81, 0, 0, 0, 37, 0, 0, 0, 0, 14, 0, 0, 26, 0, 0, 5,
36, 69, 0, 26, 0 , 63, 0, 0, 29, 0, 0, 39, 0, 0, 0, 0, 0, 0,
0, 25, 0, 0, 0, 0, 146, 0, 175, 0, 1, 0, 0, 8, 0, 22 , 0, 0,
67, 84, 0, 0, 1, 0, 0, 46, 0, 0, 27, 120, 122, 141, 0, 71,
72, 87, 88, 89, 120, 121 , 0, 71, 0, 72, 87, 88, 89, 121, 0,
121, 0, 0, 0, 0, 27, 38, 71, 72, 77, 87, 88, 89, 100 , 109,
120, 121, 122, 141, 0, 0, 0, 11, 16, 17, 40, 41, 49, 50, 55,
56, 57, 58, 70, 80, 90 , 106, 107, 0, 0, 96, 160, 0, 0, 181,
0, 0, 65, 183, 0, 13, 0, 0, 0, 0, 40, 0, 0, 105, 0, 109 , 0,
0, 0, 42, 0, 0, 0, 0, 28, 0, 28, 150, 0, 0, 0, 0, 112, 128,
```

```
0, 0, 0, 0, 0, 0, 0, 107, 0, 0 , 0, 0, 0);


DCL READ2(*) ADDR PUB INITIAL (0, 82, 83, 85, 270, 272, 271,
416, 417, 67, 84, 86, 378, 277, 311, 367, 46, 273, 379, 376,
356, 312, 335, 418, 310, 296, 297, 290, 294, 295, 10, 296,
18, 297, 66, 73, 76, 81, 325, 290, 294, 202, 62, 11, 296,
188, 297, 441, 65, 440, 442, 410, 10, 296, 297, 290, 294,
202, 479, 11, 188, 441, 65, 440, 442, 410, 59, 378, 355,
204, 60, 303, 15, 58, 64, 70, 74, 470, 201, 475, 483, 283,
203, 289, 283, 66, 73, 76, 325, 276, 382, 368, 375, 58, 64,
70, 74, 470, 201, 475, 483, 203, 75, 78, 68, 83, 85, 291,
295, 424, 425, 428, 426, 427, 410, 2, 3, 4, 394, 201, 69,
335, 7, 366, 54, 8, 44, 52, 5, 17, 407, 56, 5, 13, 17, 407,
189, 199, 200, 392, 463, 473, 437, 55, 49, 50, 324, 341,
192, 9, 193, 453, 193, 80, 197, 198, 187, 41, 408, 340, 51,
380, 381, 21, 32, 445, 280, 31, 484, 485, 359, 360, 308, 35,
40, 195, 39, 468, 30, 45, 33, 12, 190, 457, 42, 436, 42, 57,
79, 365, 34, 47, 37, 53, 38, 473, 186, 432, 433, 196, 415,
421, 434, 432, 433, 434, 191, 36, 48, 194, 462, 19, 22, 20,
22, 0, 0, 0, 0);


DCL LOOK2(*) ADDR PUB INITIAL
             (0, 6, 6, 419, 14, 14, 243, 244, 16, 23, 285,
24, 299, 25, 351, 26, 26 , 245, 246, 246, 246, 27, 247, 247,
28, 248, 248, 248, 29, 43, 420, 249, 249, 249, 61 , 250,
250, 250, 63, 251, 251, 251, 251, 71, 252, 72, 253, 253, 77,
295 , 310, 410, 410, 410, 410, 458, 335, 310, 87, 88, 89,
91, 254, 92, 255, 93, 256, 257 , 94, 97, 97, 97, 97, 97,
429, 98, 98, 98, 98, 98, 430, 99, 99, 99, 99 , 99, 431, 108,
108, 258 , 259, 259, 109, 393, 385, 116, 116, 116, 116, 435,
471, 471, 117, 472, 472, 118, 260 , 260, 260, 260, 120, 261,
121, 262, 262, 262, 122, 130, 450, 131, 459, 132, 460, 135 ,
326, 263, 263, 141, 358, 146, 264, 264, 150, 157, 447, 158,
337, 159, 336, 163, 374 , 164, 456, 165, 265, 170, 266, 171,
171, 267, 174, 268, 175, 175, 269, 177, 177, 177 , 177, 177,
177, 177, 413, 178, 178, 178, 414);


$EJECT DCL APPLY2(*) ADDR PUB INITIAL
    (0, 0, 236, 149, 136, 275, 274, 102, 363, 103, 361, 101,
209, 152, 282, 452, 281, 104, 208, 119, 287, 286, 145, 345,
345, 345, 288, 309, 345, 345, 345, 111, 293, 292, 95, 95,
95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 96,
210, 166, 301, 300, 114, 354, 331, 348, 323, 302, 322, 322,
347, 304, 349, 383, 377, 383, 139, 140, 307, 156, 305, 314,
313, 315, 173, 321, 320, 319, 316, 215, 134, 133, 227, 330,
329, 180, 409, 334, 144, 333, 327, 328, 233, 232, 123, 339,
338, 161, 344, 343, 317, 346, 318, 306, 211, 179, 353, 352,
172, 105, 229, 155, 154, 362, 364, 239, 240, 110, 185, 184,
```

```
37Ø, 369, 234, 372, 373, 371, 162, 237, 238, 113, 148, 147,
278, 455, 39Ø, 387, 467, 454, 388, 388, 469, 474, 476, 217,
393, 385, 391, 386, 222, 222, 222, 222, 222, 221, 125, 124,
223, 389, 395, 115, 22Ø, 115, 115, 115, 115, 115, 115, 219,
115, 115, 115, 115, 115, 218, 4Ø6, 143, 129, 224, 411, 224,
412, 226, 461, 451, 4Ø5, 48Ø, 126, 127, 482, 481, 128, 481,
225, 181, 213, 214, 212, 183, 242, 241, 16Ø, 439, 423, 422,
168, 167, 438, 151, 231, 449, 448, 4Ø2, 384, 4Ø3, 138, 137,
396, 235, 444, 443, 4ØØ, 23Ø, 182, 465, 464, 228, 228, 142,
4Ø1, 1ØØ, 176, 2Ø7, 2Ø6, 2Ø5, 397, 1Ø6, 399, 112, 169, 153,
478, 477, 398, 216, 9Ø, 1Ø7);


DCL INDEX1(*) ADDR PUB INITIAL
                (Ø, 1, 4, 5, 6, 22, 7, 9, 9, 13, 14, 43, 43,
43, 12Ø, 52, 43, 43, 24, 15 , 16, 17, 9, 83, 71, 68, 12Ø,
73, 25, 25, 18, 84, 13, 19, 2Ø, 21, 22, 52, 114, 25, 43, 43
, 43, 23, 24, 24, 24, 3Ø, 3Ø, 43, 43, 25, 3Ø, 42, 25 . 43,
43, 43, 43, 52, 3Ø, 25, 3Ø, 25 , 58, 59, 66, 67, 68, 69, 43,
93, 93, 7Ø, 84, 71, 72, 73, 83, 84, 43, 85, 89, 9Ø, 67 , 91,
92, 93, 93, 93, 43, 1Ø2, 1Ø3, 1Ø4, 1Ø5, 1Ø7, 59, 1Ø9, 1Ø9,
1Ø9, 114, 115, 116 , 117, 118, 119, 43, 43, 12Ø, 73, 122,
124, 141, 125, 127, 128, 132, 128, 128, 136, 73, 93 , 73,
24, 137, 138, 139, 14Ø, 141, 142, 143, 144, 145, 146, 147,
148, 149, 15Ø, 152 , 154, 155, 73, 156, 157, 159, 16Ø, 1Ø5,
161, 162, 163, 25, 165, 166, 168, 17Ø, 171 , 172, 174, 175,
175, 59, 176, 178, 18Ø, 181, 188, 182, 183, 185, 187, 188,
188, 19Ø , 192, 188, 188, 194, 196. 2Ø3, 2Ø6, 2Ø7, 43, 2Ø9,
59, 211, 213, 1, 4, 7, 9, 11, 13, 15 , 18, 22, 25, 29, 31,
35, 39, 44, 46, 49, 51, 56, 58, 59, 6Ø, 61, 63, 65, 67, 69,
75, 81 , 87, 9Ø, 93, 95, 1ØØ, 1Ø3, 1Ø6, 111, 113, 117, 119,
121, 123, 125, 128, 13Ø, 133, 135 , 137, 139, 141, 143, 145,
147, 15Ø, 152, 155, 163, 332, 446, 332, 4Ø4, 466, 342, 342 ,
342, 4Ø4, 4Ø4, 4Ø4, 298, 284, 35Ø, 357, 332, 4Ø4, 4Ø4, 4Ø4,
4Ø4, 4Ø4, 466, 279, 279 , 279, 279, 279. 1, 1, 1, 2, 3, 3,
4, 5, 7, 12, 12, 13, 13, 14, 18, 18, 19, 19, 2Ø, 22, 23 ,
23, 23, 23, 23, 32, 34, 34, 51, 51, 52, 52, 53, 55, 56, 56,
56, 61, 61, 61, 65, 72, 72 , 73, 73, 74, 74, 74, 74, 76, 77,
77, 78, 8Ø, 81, 82, 83, 83, 83, 85, 85, 86, 86, 88, 88 , 89,
93, 93, 95, 95, 97, 98, 98, 1ØØ, 1ØØ, 1Ø1, 1Ø3, 1Ø4, 1Ø5,
1Ø6, 1Ø7, 1Ø7, 1Ø8, 1Ø8 , 1Ø9, 111, 111, 112, 112, 113, 113,
114, 114, 114, 114, 116, 118, 118, 12Ø, 121, 121 , 123, 123,
123, 123, 125, 125, 126, 129, 129, 13Ø, 13Ø, 132, 133, 135,
136, 136, 136 , 141, 141, 149, 149, 151, 157, 159, 16Ø, 16Ø,
16Ø, 16Ø, 16Ø, 16Ø, 16Ø, 16Ø, 16Ø, 16Ø , 161, 162, 162, 162,
162, 177, 178, 178, 179, 179, 196, 196, 196, 196, 196, 196,
196 , 197, 197, 2ØØ, 2ØØ, 2ØØ, 2ØØ, 2ØØ, 2Ø1, 2Ø1, 2Ø1, 2Ø3,
2Ø3, 2Ø3, 2Ø4, 2Ø4, 2Ø4, 2Ø4 , 2Ø4, 2Ø4, 2Ø4, 2Ø4, 2Ø7, 2Ø7,
2Ø9, 21Ø, 21Ø, 211, 211, 212, 212, 214, 215, 217, 217 , 219,
219, 22Ø, 221, 221, 221, 223, 224, 225, 225, 226, 226, 228,
231, 232, 233, 233 , 234, 237, 238, 239, 24Ø, 24Ø, 241, 242,
243, 245, 246, 247, 248);
```

```
$EJECT DCL INDEX2 (*) BYTE PUB DATA(Ø, 3, 1, 1, 1, 1, 2, 4,
4, 1, 1, 9, 9, 9, 2, 6, 9, 9, 1, 1, 1, 1, 4, 1, 1, 1 , 2,
1Ø, 5, 5, 1, 1, 1, 1, 1, 1, 1, 6. 1, 5. 9, 9, 9, 1, 1, 1, 1,
12, 12, 9, 9, 5, 12, 1, 5, 9, 9, 9 , 9, 6, 12, 5, 12, 5, 1,
7, 1, 1, 1, 1, 9, 9, 9, 1, 1, 1, 1, 1Ø, 1, 1, 9, 4, 1, 1, 1,
1, 1, 9, 9 , 9, 9, 1, 1, 1, 2, 2, 7, 5, 5, 5, 1, 1, 1, 1, 1,
1, 9, 9, 2, 1Ø, 2, 1, 1, 2, 1, 4, 4, 3, 3, 1, 1Ø, 9 , 1Ø, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1Ø, 1, 2,
1, 1, 2, 1, 1, 2, 5, 1, 2, 2, 1 , 1, 2, 1, 1, 1, 7, 2, 2, 1,
1, 1, 1, 2, 2, 1, 1, 2, 2, 2, 1, 2, 2, 7, 3, 1, 2, 9, 2, 7,
2, 2, 3, 3, 2 , 2, 2, 2, 3, 4, 3, 4, 2, 4, 4, 5, 2, 3, 2, 5,
2, 1, 1, 1, 2, 2, 2, 2, 6, 6, 6, 3, 3, 2, 5, 3, 3, 5, 2 , 4,
2, 2, 2, 2, 3, 2, 3, 2, 2, 2, 2, 2, 2, 2, 3, 2, 3, 8, 4, 14,
16, 26, 27, 28, 29, 61, 63, 71 , 72, 77, 91, 92, 93, 94,
1Ø8, 1Ø9, 12Ø, 121, 122, 141, 15Ø, 165, 17Ø, 171, 174, 175,
3 , 3, 3, 5, Ø, 2, Ø, Ø, 5, Ø, 2, Ø, 2, Ø, Ø, 2, Ø, 2, 2, Ø,
Ø, 1, Ø, 1, Ø, Ø, Ø, Ø, Ø, 2, Ø, 2. 2, Ø, Ø , Ø, Ø, Ø, 2, 2,
Ø, Ø, 2, Ø, 1, Ø, Ø, Ø, Ø, 5, Ø, 2, Ø, Ø, 2, Ø, Ø, 2, Ø, Ø,
2, 2, Ø, Ø, 2, Ø, 4, 3, Ø, 2 , 1, 3, Ø, Ø, 2, Ø, 2, Ø, 2, 1,
Ø, 2, Ø, 2, 2, Ø, 2, Ø, Ø, 1, 2, 1, 1, 1, 1, Ø, 1, 4, 1, Ø,
2, Ø, 1, 1 , 1, Ø, 2, 2, Ø, 2, 3, 6, 1, Ø, Ø, Ø, Ø, 1, 3, Ø,
1, 3, 2, Ø, 1, Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø, 2, Ø. 1 , 3. 2,
Ø, Ø, 2, Ø, 2, Ø, 1, 1, 1, Ø, Ø, Ø, Ø, 2, Ø, Ø, Ø, Ø, Ø, Ø,
1, 2, Ø, Ø, Ø, Ø, 3, 2, Ø, 1, Ø, Ø , Ø, Ø, 2, 2, Ø, Ø, Ø, 2,
Ø, 2, 1, 2, Ø, 2, Ø, 3, Ø, Ø, 2, 4, 2, 2, Ø, 2, Ø, 1, 1, 3,
Ø, Ø, 2, Ø, 3, Ø , 5, 2, 2, Ø, Ø, Ø, 3, Ø, Ø, Ø);

END TABLES;
```

```
$PAGEWIDTH(80) TITLE('SYMBOL - SYMBOL TABLE ROUTINES')
 SYMBOL:DO;
 DECLARE LIT LITERALLY 'LITERALLY',
        DCL LIT 'DECLARE',
        POS LIT '0',
        NEG LIT '1', PUB LIT 'PUBLIC', EXT LIT 'EXTERNAL',
        PROC LIT 'PROCEDURE',
        TRUE LIT '1',
        ADDR LIT 'ADDRESS',
        FALSE LIT '0',
        BUILT$IN$FUNC LIT '0DH'; DCL
        IDENTSIZE LIT '32', /* MAX IDENTIFIER SIZE + 1 */
        VARCSIZE LIT '100', /* SIZE OF VARC STACK*/
        PSTACKSIZE LIT '48', /* SIZE OF PARSE STACKS */
        HASHTBLSIZE LIT '128', /* SIZE OF HASHTABLE */
        HASHMASK LIT '127', /* HASHTABLE SIZE -1 */
        MAXINT LIT '32767', /* MAX INTEGER VALUE */
        BCDSIZE LIT '8', /* BYTES USED FOR BCD VALUES */
        MAX$NEST LIT '3', /* MAX LEVEL OF NESTS FOR TYPES */
        MAX$ARRY$DIM LIT '5', /* MAX ARRY DIMENSIONS */
        FORMMASK LIT '7', /* USED TO DETERMINE FORM TYPE */
/* FORM ENTRIES */
        CONS$ENTRY LIT '1',
        TYPE$ENTRY LIT '2',
        VAR$ENTRY LIT '3',
        FUNC$ENTRY LIT '5', /* NUMBER TYPES */
        INTEGER$TYPE LIT '1',
        SIGNED$EXPON LIT '4',
        REAL$TYPE LIT '2',
        PARM LIT '67',
        LODI LIT '79',
        PARMV LIT '68';
        /* MANY OF THE FOLLOWING VARIABLES CAN BE REPLACED
BY MAKING
            USE OF THE PARALLEL PARSE STACKS */
 DCL
        FORM BYTE EXT,
        EXPON BYTE EXT,
        VECPTR BYTE EXT,
        TYPENUM BYTE EXT,
        CONST$PTR BYTE EXT,
        STARTBDOS ADDR EXT, /*ADDR OF PTR TO TOP OF BDOS*/
        TYPE$LOCT ADDR EXT,
        VAR$PTR BYTE EXT,
        VAR$BASE1(10) ADDR EXT,
        ALLC$QTY ADDR EXT,
        CONST$INDX BYTE EXT,
        LOOKUP$ADDR ADDR EXT,
        CONST$VALUE(16) BYTE EXT,
        CONST$PN$BASE(4) BYTE EXT,
        CONST$PN$PTR BYTE EXT,
```

```
        CONST$PN$SIZE(4) BYTE EXT,
        CUR$OFST(MAX$NEST) ADDR EXT,
        /* CASE STATEMENT VARIABLES */
        CONST$NUM$TYPE(4) BYTE EXT; /*** GLOBAL VARIABLES
***/
 DCL BCDNUM(BCDSIZE) BYTE EXT,
        SCOPE(10) ADDR EXT,
        SCOPE$NUM BYTE EXT,
        TEMPBYTE BYTE EXT,
        TEMPADDR ADDR EXT,
        TEMPADDR1 ADDR EXT,
        PRV$SBT$ENTRY ADDR EXT;
 DCL
              /* COMPILER TOGGLES */

              /* COUNTERS */
         LABLCOUNT ADDR EXT, /* COUNTS NUMBER OF LABELS */
         ALLOC$ADDR ADDR EXT, /* COUNTS PRT ENTRIES */

              /* FLAGS USED DURING CODE GENERATION */
         READPARMS BYTE EXT, /* READING ACTUAL PARAMETERS */
         PRESENT BYTE EXT, /* IDENTIFIER IS IN SYMBOL TABLE
*/
         SIGN$FLAG BYTE EXT, /* SET WHEN SIGN PRECEDES ID */

              /* GLOBAL VARIABLES USED BY THE SCANNER */
         HASHCODE BYTE EXT, /* HASH VALUE OF CURRENT TOKEN
*/

              /* GLOBAL VARIABLES USED IN SYMBOL TABLE
OPERATIONS */
         BASE ADDR EXT, /* BASE LOCATION OF ENTRY */
         HASHTABLE(HASHTBLSIZE) ADDR EXT, /* HASHTABLE ARRAY
*/
         SBTBLTOP ADDR EXT, /* HIGHEST LOCATION OF SYMBOL
TABLE */
         SBTBL ADDR EXT, /* CURRENT TOP OF SYMBOL TABLE */
         APTRADDR ADDR EXT, /* UTILITY VARIABLE TO ACCESS
SBTBL */
         ADDRPTR BASED APTRADDR ADDR, /* CURRENT 2 BYTES
POINTED AT */
         (BYTEPTR BASED APTRADDR)(1) BYTE, /* CURRENT BYTE
POINTED AT */
         PRINTNAME ADDR EXT, /* SET PRIOR TO LOOKUP OR ENTER
*/
         SYMHASH BYTE EXT, /* HASH VALUE OF AN IDENTIFIER */
         LAST$SBTBL$ID ADDR EXT, /* HOLD PREVIOUS BASE
LOCATION */
         PARAMNUMLOC ADDR EXT, /* STORES POINTER TO PARAM
LISTING */
         SBTBLSCOPE ADDR EXT, /* BASE OF LAST ENTRY IN
PREVIOUS BLOCK*/
         SP BYTE EXT,
         MP BYTE EXT,
```

```
        PARMNUMLOC(PSTACKSIZE) ADDR EXT,
        PARAMNUM BYTE EXT,
        PRTADDR(PSTACKSIZE) ADDR EXT,
        EXPRESS$STK(PSTACKSIZE) BYTE EXT,
        FORM$FIELL(PSTACKSIZE) BYTE EXT,
        VAR(PSTACKSIZE) BYTE EXT,
        VARC(VARCSIZE) BYTE EXT,
        BASE(PSTACKSIZE) BYTE EXT;

/* DECLARE EXTERNAL PROCEDURES, FOUND IN SYSRTS */

GENERATE: PROC(OBJCODE) EXT;
    DCL OBJCODE BYTE;
 END GENERATE;

ERROR: PROC(ERRCODE) EXT;
    DCL ERRCODE ADDR;
 END ERROR;

MOVE: PROC(SOURCE,DESTIN,L) EXT;
    DCL (SOURCE,DESTIN) ADDR;
    DCL L BYTE;
 END MOVE;

MON3: PROC EXT;
 END MON3;

GENADDR: PROC(A,B) EXT;
    DCL A BYTE,B ADDR;
 END GENADDR;

  /*************************************************
   * SET$ADDRESS$PTR - THIS PROCEDURE SETS A *
   * POINTER TO A SPECIFIC LOCATION IN THE *
   * SYMBOL TABLE. *
   *************************************************/
SETADDRPTR: PROC(OFFSET) PUB;
   DCL OFFSET BYTE;
   APTRADDR = BASE + OFFSET;
 END SETADDRPTR;
  /*************************************************
   * SET$PAST$PRINTNAME - THIS PROCEDURE SETS *
   * APTRADDR TO A LOCATION IN A SYMBOL TABLE *
   * ENTRY THAT IS PAST THE ENTRY'S PRINTNAME *
   * (WHICH IS OF VARIABLE LENGTH). *
   *************************************************/
SET$PAST$PN: PROC(OFFSET) PUB;
   DCL OFFSET BYTE;
   CALL SETADDRPTR(6);
   CALL SETADDRPTR(BYTEPTR(0) + OFFSET);
 END SET$PAST$PN;
  /*************************************************
   * CALC$VARC - THIS PROCEDURE DETERMINES THE *
   * LOCATION OF AN IDENTIFIER PRINTNAME. *
```

```
    ****************************************************************/
CALC$VARC: PROC(A) ADDR PUB;
    DCL A BYTE;
    RETURN VAR(A) + .VARC;
 END CALC$VARC;
  /***************************************************************
   * SET$LOOKUP - THIS PROCEDURE IS UTILIZED TO *
   * FIND THE HASH VALUE OF AN IDENTIFIER. *
   ****************************************************************/
SETLOOKUP: PROC(A) PUB;
    DCL A BYTE;
    PRINTNAME = CALC$VARC(A);
    SYMHASH = HASH(A); /* HASHCODE OF PN */
 END SETLOOKUP;
  /***************************************************************/
  /* ENTER$LINKS - THIS PROCEDURE ENTERS IN THE */
  /* NEXT FOUR BYTES OF THE SYMBOL TABLE THE */
  /* COLLISION FIELD AND THE PREVIOUS SYMBOL */
  /* TABLE ENTRY ADDRESS FIELD FOR THE NEXT */
  /* SYMBOL TABLE ENTRY. ( BOTH IN ADDRESS VAR ) */
  /***************************************************************/
ENTER$LINKS: PROC PUB;
    BASE, APTRADDR, SBTBLSCOPE = SBTBL;
    SCOPE(SCOPE$NUM) = SBTBL;
    ADDRPTR = HASHTABLE(SYMHASH);
    CALL SETADDRPTR(2);
    ADDRPTR = PRV$SBT$ENTRY;
    PRV$SBT$ENTRY = SBTBL;
    HASHTABLE(SYMHASH) = BASE;
 END ENTER$LINKS;
  /***************************************************************
   * CHECK$PRINT$NAME - THIS PROCEDURE DOES A *
   * CHARACTER TO CHARACTER COMPARISON BETWEEN *
   * THE CURRENTLY RECOGNIZED IDENTIFIER AND *
   * SYMBOL TABLE ENTRIES OF THE SAME HASH VALUE.*
   ****************************************************************/
CHK$PRT$NAME: PROC(A) BYTE PUB;
    /* A IS OFFSET FROM BASE TO PRINTNAME */
    DCL(N BASED PRINTNAME)(1) BYTE;
    DCL (LEN,A) BYTE;
    CALL SETADDRPTR(A);
    IF ( LEN := BYTEPTR(0) ) =N(0) THEN
    DO WHILE (BYTEPTR(LEN)=N(LEN));
      IF ( LEN := LEN-1 ) = 0 THEN
        RETURN TRUE;
    END;
    RETURN FALSE;
 END CHK$PRT$NAME;
  /***************************************************************/
  /* LOOKUP$PRINTNAME$IDENTITY - THIS PROCEDURE */
  /* IS PASSED THE LOCATION OF AN IDENTIFIER IN */
  /* THE PRODUCTION RULE, AND ITS TARGET ENTRY */
  /* TYPE. IF THE IDENTIFIER IS FOUND WITH THE */
  /* CORRECT TYPE THE PROCEDURE RETURN TRUE, */
```

```
  /* ELSE FALSE IS RETURNED. */
  /**************************************************/
LOOKUP$PN$ID: PROC(A,ID$ENTRY) BYTE PUB;
    DCL (A,ID$ENTRY) BYTE;
    CALL SETLOOKUP(A);
    BASE = HASHTABLE(SYMHASH);
    DO WHILE BASE <> 0;
      CALL SETADDRPTR(4);
      IF (( BYTEPTR(0) AND FORMMASK ) = ID$ENTRY ) THEN
        IF CHK$PRT$NAME(6) THEN
          IF ((BASE < SCOPE(0)) OR (BASE >=
SCOPE(SCOPE$NUM-1))
          OR ((ID$ENTRY = TYPE$ENTRY) AND (BASE <
SCOPE(SCOPE$NUM)))
          THEN DO;
            LOOKUP$ADDR=BASE;
            RETURN TRUE;
          END;
      CALL SETADDRPTR(0);
      BASE = ADDRPTR;
    END;
    RETURN FALSE;
 END LOOKUP$PN$ID;
  /**************************************************/
  /* LIMITS - THIS PROCEDURE ENSURES THAT THE */
  /* SYMBOL TABLE ENTRY ABOUT TO BE ENTERED */
  /* WILL NOT EXCEED THE UPPER LIMIT OF THE */
  /* AVAILABLE SYMBOL TABLE ADDRESSES. */
  /* THE PARAMETER IS THE BYTECOUNT OF THE */
  /* ENTRY TO BE ENTERED. */
  /**************************************************/
 LIMITS: PROC(COUNT) PUB;
    DCL COUNT BYTE;
    IF SBTBLTOP <= (SBTBL + COUNT) THEN
    DO;
      CALL ERROR('TO');
      CALL MON3;
    END;
 END LIMITS;
  /**************************************************/
  /* ENTER$PRINTNAME$IDENTITY - THIS PROCEDURE */
  /* LOADS THE SYMBOL TABLE WITH THE FOLLOWING: */
  /* 1. COLLISION FIELD */
  /* 2. PREVIOUS SYMBOL TABLE ENTRY ADDRESS */
  /* 3. FORM OF ENTRY ( PRESET BYTE "FORM" ) */
  /* 4. THE LENGTH OF THE PRINTNAME IN ONE BYTE*/
  /* 5. THE PRINTNAME CHARACTERS */
  /* PARAMETER: PRINTNAME IS SET PRIOR TO CALL. */
  /**************************************************/
ENTER$PN$ID:PROC PUB;
     DCL I BYTE;
    DCL (N BASED PRINTNAME)(1) BYTE;
    CALL LIMITS(I:=N(0)+7);
    CALL ENTER$LINKS;
```

```
    CALL SETADDRPTR(4);
    BYTEPTR(0)= FORM;
    CALL SETADDRPTR(5);
    BYTEPTR(0)= SYMHASH;
    CALL SETADDRPTR(6);
    BYTEPTR(0)=N(0);
    CALL MOVE(PRINTNAME+1,SBTBL+7,N(0));
    LAST$SBTBL$ID = SBTBL;
    SBTBL=SBTBL+I;
 END ENTER$PN$ID;
  /*************************************************/
  /* ENTER$VARIABLE$IDENTITY - THIS PROCEDURE */
  /* CALLS ENTER$PN$ID TO LOAD THE SYMBOL TABLE */
  /* ENTRY CURRENTLY BEING SCANNED. IT ALSO */
  /* GENERATES THE ENTRY'S "FORM" BY PERFORMING */
  /* A BOOLEAN 'OR' OPERATION ON THE ID$ENTRY */
  /* AND THE PARAMETER "A". */
  /*************************************************/
ENTER$VAR$ID: PROC(A,B,ID$ENTRY) PUB;
    DCL (A,B,ID$ENTRY) BYTE;
    IF LOOKUP$PN$ID(B,ID$ENTRY) THEN
    DO;
      PRESENT = TRUE;
      RETURN;
    END;
    /* ELSE ENTER VAR NAME */
    PRESENT = FALSE;
    FORM = A OR ID$ENTRY;
    CALL ENTER$PN$ID;
    IF ID$ENTRY = VAR$ENTRY THEN
    DO;
      CALL LIMITS(4);
      VAR$BASE1(VAR$PTR) = SBTBL;
      SBTBL = SBTBL + 4;
    END;
 END ENTER$VAR$ID;
  /*************************************************/
  /* SET$LABEL - THIS PROCEDURE ASSIGNS A LABEL */
  /* TO THE CURRENT DECLARED LABEL AND INCREMENT*/
  /* THE LABELCOUNT ( NEXT TO ASSIGN ). */
  /*************************************************/
SET$LABEL: PROC PUB;
    ADDRPTR=LABLCOUNT;
    LABLCOUNT=LABLCOUNT+1;
 END SET$LABEL;
  /*************************************************/
  /* ENTER$LABEL - THIS PROCEDURE LOADS A LABEL */
  /* ENTRY INTO THE SYMBOL TABLE. SYMHASH AND */
  /* PRINTNAME MUST BE SET PRIOR TO CALLING */
  /*************************************************/
ENTER$LABEL: PROC PUB;
    CALL LIMITS(2);
    APTRADDR = SBTBL;
    CALL SET$LABEL;
```

```
    SBTBL = SBTBL+2;
 END ENTER$LABEL;
  /**************************************************
   * ALTER$PRT$LOCATIONS - THIS PROCEDURE RE- *
   * ALLOCATES PRT LOCATIONS FOR ALL FUNCTIONS *
   * AND FORWARD PROCEDURES AND THEIR ASSOCIATED*
   * FORMAL PARAMETERS. *
   **************************************************/
ALTER$PRT$LOC: PROC PUB;
    DCL (I,P) BYTE;
    CALL SET$PAST$PN(7);
    P = BYTEPTR(0);
    PARAMNUMLOC = APTRADDR;
    DO I = 1 TO P;
      CALL SET$PAST$PN(8);
      APTRADDR = ADDRPTR + ((I-1)*3);
      DO CASE (SHR(BYTEPTR(0),3) AND FORMMASK);
        ALLC$QTY = 1; /* SCALAR */
        ALLC$QTY = 2; /* INTEGER */
        ALLC$QTY = 8; /* REAL */
        ALLC$QTY = 1; /* CHAR */
        ALLC$QTY = 1; /* BOOLEAN */
      END; /* OF CASE */
      APTRADDR = APTRADDR + 1;
      ADDRPTR = ALLOC$ADDR;
      APTRADDR = TEMPADDR1;
      APTRADDR = APTRADDR + 6;
      APTRADDR = APTRADDR + 1 + BYTEPTR(0);
      ADDRPTR = ALLOC$ADDR;
      ALLOC$ADDR = ALLOC$ADDR + ALLC$QTY;
      TEMPADDR1 = APTRADDR + 4;
    END;
 END ALTER$PRT$LOC;
  /**************************************************
   * ENTER$SUBROUTINE - THIS PROCEDURE LOADS A *
   * SUBROUTINE ENTRY IN THE SYMBOL TABLE. THE *
   * PARAMETER NUMBER LOCATION IS STORED AND THE *
   * SCOPE LEVEL IS INCRIMENTED BY ONE. *
   **************************************************/
ENTER$SUBRTN: PROC(A,B,ID$ENTRY) PUB;
    DCL (A,B,ID$ENTRY) BYTE;
    CALL ENTER$VAR$ID(0, SP, ID$ENTRY) ;
    IF NOT PRESENT THEN
    DO;
      CALL LIMITS(4);
      PARAMNUMLOC = SBTBL;
      SBTBL = SBTBL + 3;
      CALL SET$PAST$PN(10);
      ADDRPTR = ALLOC$ADDR; ALLOC$ADDR = ALLOC$ADDR + 2;
      CALL SET$PAST$PN(14);
      ADDRPTR = LABLCOUNT;
      LABLCOUNT = LABLCOUNT + 2;
      SBTBL = SBTBL + 6;
      IF ID$ENTRY = FUNC$ENTRY THEN
```

```
      DO;
        SBTBL = SBTBL + 1;
      END;
    END;
    ELSE DO; /* FORWARD FUNCTION */
      CALL SET$PAST$PN(14);
      IF ID$ENTRY = FUNC$ENTRY THEN TEMPADDR1 = APTRADDR +
3;
      ELSE TEMPADDR1 = APTRADDR + 2;
      CALL SET$PAST$PN(10);
      ADDRPTR = ALLOC$ADDR;
      ALLOC$ADDR = ALLOC$ADDR + 2;
      CALL ALTER$PRT$LOC;
    END;
    PARMNUMLOC(MP) = BASE;
    SCOPE(SCOPE$NUM := SCOPE$NUM+1) = SBTBL;
 END ENTER$SUBRTN;
  /*********************************************************/
  /* LOOKUP$ONLY - THIS PROCEDURE IS PASSED THE */
  /* POSITION OF A IDENTIFIER JUST SCANNED IN */
  /* THE CURRENT PRODUCTION ( SP,MP,MPP1 ) AND */
  /* RETURNS TRUE IF THE IDENTIFIER IS FOUND IN */
  /* THE SYMBOL TABLE. */
  /*********************************************************/
LOOKUP$ONLY: PROC(A) BYTE PUB;
    DCL A BYTE;
    CALL SETLOOKUP(A);
    BASE=HASHTABLE(SYMHASE);
    DO WHILE BASE <> 0;
      IF CHK$PRT$NAME(6) THEN
      DO;
        LOOKUP$ADDR=BASE;
        RETURN TRUE;
      END;
      ELSE DO;
        CALL SETADDRPTR(0);
        BASE=ADDRPTR;
        END;
      END;
    RETURN FALSE;
 END LOOKUP$ONLY;
 /****************************************/
 /* THIS PROCEDURE CONVERTS A REAL */
 /* NUMBER IN THE PROGRAM TO A BCD */
 /* REPRESENTATION. */
 /****************************************/
 CONVRTBCD: PROC(A,B) PUB; /* A=SP/MP/MPP1, B=POS/NEG */
    DCL (I,J,DFLAG,EFLAG,SFLAG,A,B) BYTE;
    DCL (N BASED PRINTNAME)(1) BYTE;
    DCL (EXPONLOOP,EXPSIGNLOOP) LABEL;
    CALL SETLOOKUP(A);
     /* INITIALIZE VARIABLES */
    SFLAG=FALSE; EFLAG=TRUE; DFLAG=TRUE; I=1;
    DO J=0 TO 7; BCDNUM(J)=0; END;
```

```
J=Ø; EXPON=64; /* E+ØØ */
/* REMOVE LEADING ZEROS */
DO WHILE ((N(I) - 'Ø') = Ø);
   I=I+1;
   IF I=(N(Ø)+1) THEN GOTO EXPONLOOP;
END;
/* LOAD BCDNUM WITH SIGNIFICANT DIGITS */
DO WHILE ((N(I) - 'Ø' ) <= 9 OR N(I) = '.');
   IF N(I) = '.' THEN
   DO; EFLAG=FALSE;
      IF I=N(Ø) THEN GOTO EXPONLOOP;
      I = I + 1;
   END;
   ELSE
   DO;
      DO WHILE J = Ø AND DFLAG AND (N(I) - 'Ø') = Ø;
         EXPON = EXPON-1;
         IF I = N(Ø) THEN GOTO EXPONLOOP;
         I = I + 1 ;
      END;
      IF J = ( BCDSIZE-1 ) THEN GOTO EXPONLOOP;
      IF DFLAG THEN /* FIRST BCD PAIR */
      DO;
         BCDNUM(J)=ROL((N(I)-'Ø'),4);
         DFLAG=FALSE; I= I+1;
         IF EFLAG THEN EXPON=EXPON+1;
      END;
      ELSE
      DO;
         BCDNUM(J)=BCDNUM(J)+(N(I)-'Ø');
         J = J + 1; I = I + 1;
         DFLAG=TRUE; IF EFLAG THEN EXPON=EXPON+1;
      END;
      IF I=(N(Ø)+1) THEN GOTO EXPONLOOP;
   END;
END;
EXPONLOOP:
IF N(I) = 'E' THEN EFLAG = FALSE;
IF I = (N(Ø)+1) THEN GOTO EXPSIGNLOOP;
IF EFLAG THEN
DO;
   DO WHILE N(I) <> '.';
      EXPON = EXPON + 1;
      I = I + 1;
   END;
   I = I + 1;
END;
DO WHILE I < (N(Ø)+1) AND (N(I)-'Ø') <= 9 ;
   I = I + 1;
END;
IF TYPENUM = REALTYPE THEN GOTO EXPSIGNLOOP;
/* N(I) = E */ I = I+1;
IF TYPENUM = SIGNED$EXPON THEN
DO;
```

```
      IF N(I) = 2DH THEN SFLAG = TRUE;
      I = I + 1 ;
    END;
    IF I = N(0)+1 THEN
    DO;
      CALL ERROR('EE');
      RETURN;
    END;
    DFLAG = 0;
    DO J = I TO N(0);
      DFLAG = (DFLAG*10)+(N(J)-'0');
    END;
    IF SFLAG THEN /* EXPONENT CALCULATION */
       EXPON = EXPON-DFLAG;
    ELSE EXPON = EXPON + DFLAG;
    EXPSIGNLOOP:
    BCDNUM(BCDSIZE-1)=ROL(B,7); /* SIGN OF NUMBER */
    IF EXPON > 127 THEN
    DO;
      CALL ERROR('EE');
      RETURN;
    END;
    ELSE BCDNUM(BCDSIZE-1)=BCDNUM(BCDSIZE-1)+EXPON;
 END CONVRTBCD;
  /*****************************************************/
  /* CONVERTI - THIS PROCEDURE IS PASSED "A", THE*/
  /* LOCATION OF A CONSTANT IN THE PRODUCTION */
  /* AND "B" THE 'SIGN' OF THE INTEGER. THE */
  /* FUNCTION GENERATES A SIGNED 16 BIT REPRE- */
  /* SENTATION OF THE NUMBER AND RETURNS IT IN */
  /* AN ADDRESS VARIABLE. */
  /*****************************************************/
CONVERTI: PROC(A,B) ADDRESS PUB;
    DCL (I,A,B) BYTE;
    DCL (N BASED PRINTNAME)(1) BYTE;
    DCL NUM ADDR;
    CALL SETLOOKUP(A); NUM=0;
    DO I=1 TO N(0);
      IF (MAXINT/10) >= NUM THEN
      DO;
        IF (MAXINT/10) = NUM AND (N(I)-'0') > 7 THEN
        DO;
          CALL ERROR('IE');
          RETURN NUM;
        END;
        NUM=(NUM*10)+(N(I)-'0');
      END;
      ELSE DO;
        CALL ERROR('IE');
        RETURN NUM;
      END;
    END;
    IF B = POS THEN RETURN NUM;
    IF NUM = MAXINT THEN
```

```
    DO;
      CALL ERROR('IF');
      RETURN NUM;
    END;
    RETURN ( - NUM);
 END CONVERTI;
  /*****************************************************/
  /* CONVERT$CONSTANT - THIS PROCEDURE IS CALLED */
  /* WITH TYPENUM SET BY THE CALLER. THE NUMBER */
  /* MUST BE POINTED TO BY "SP" IN THE PRODUC- */
  /* TION. THE PROCEDURE RETURNS WITH "CONST$ */
  /* NUM$TYPE" AND "CONST$VALUE" SET WITH THE */
  /* NUMBER IN ITS INTERNAL FORM. */
  /*****************************************************/
CONVRT$CONST: PROC(A) PUB; /* A=POS,NEG */
    DCL A BYTE,INT$ADDR ADDR;
    IF TYPENUM = INTEGER$TYPE THEN
    DO;
      INT$ADDR=CONVERTI(SP,A);
      CONST$NUM$TYPE(CONST$PTR)=INTEGER$TYPE;
      CONST$PTR=CONST$PTR+1;
      CALL MOVE(.INT$ADDR,.CONST$VALUE(CONST$INDX),2);
      CONST$INDX=CONST$INDX+2;
    END;
    ELSE DO;
      CALL CONVRTBCD(SP,A);
      CONST$NUM$TYPE(CONST$PTR)=REAL$TYPE;
      CONST$PTR=CONST$PTR+1;
      CALL MOVE(.BCDNUM,.CONST$VALUE(CONST$INDX),BCDSIZE);
      CONST$INDX=CONST$INDX+BCDSIZE;
    END;
 END CONVRT$CONST;
  /*****************************************************/
  /* ENTER$CONSTANT$NUMBER - AFTER THE NEXT ENTRY*/
  /* HAS HAD ITS LINKS ENTERED INTO THE SYMBOL */
  /* TABLE, THIS PROCEDURE ENTERS THE CONSTANT */
  /* VALUE INTO THE SYMBOL TABLE AND SET THE */
  /* ENTRY'S "FORM" TO THE APPROPRIATE TYPE. */
  /*****************************************************/
ENTR$CONS$NUM: PROC PUB;
    CONST$PTR=CONST$PTR-1;
    IF CONST$NUM$TYPE(CONST$PTR)= INTEGERTYPE THEN
    DO;
      CALL SETADDRPTR(4); BYTEPTR(0)=8 OR CONS$ENTRY;
      CALL LIMITS(2); CONST$INDX=CONST$INDX-2;
      CALL MOVE(.CONST$VALUE(CONST$INDX),SBTBL,2);
      SBTBL=SBTBL+2;
    END;
    ELSE DO;
      CALL SETADDRPTR(4); BYTEPTR(0)=10H OR CONS$ENTRY;
      CALL LIMITS(BCDSIZE); CONST$INDX=CONST$INDX-BCDSIZE;
      CALL MOVE(.CONST$VALUE(CONST$INDX),SBTBL,BCDSIZE);
      SBTBL=SBTBL+BCDSIZE;
    END;
```

```
 END ENTR$CONS$NUM;
  /*****************************************************/
  /* ENTER$STRING - AFTER THE "LINKS" AND "FORM" */
  /* ARE ENTERED INTO THE SYMBOL TABLE, THIS */
  /* PROCEDURE LOADS ANY IDENTIFIER ALONG WITH */
  /* ITS LENGTH. (USED WITH CONSTANT STRINGS */
  /* AND CONSTANT IDENTIFIERS ) */
  /*****************************************************/
ENTER$STRING: PROC(A) PUB;
    DCL (N BASED PRINTNAME)(1) BYTE;
    DCL A BYTE;
    CALL SETLOOKUP(A);
    CALL LIMITS(N(0)+1);
    CALL MOVE(PRINTNAME,SBTBL,(N(0)+1));
    SBTBL=SBTBL+(N(0)+1);
 END ENTER$STRING;
  /*****************************************************
   * ENTER$CONSTANT$ID - THIS PROCEDURE ENTERS *
   * THE FORM FIELD OF A CONSTANT ENTRY INTO *
   * THE SYMBOL TABLE. *
   *****************************************************/
ENTR$CONS$ID: PROC(A,B) PUB; /* A=POS/NEG , B=MP/MPP1/SP */
    DCL (A,B,C) BYTE;
    C=ROL(A,6);
    CALL SETADDRPTR(4); BYTEPTR(0)=C OR CONS$ENTRY;
    CALL ENTER$STRING(SP);
    CONST$PN$PTR=CONST$PN$PTR-1;
    CONST$INDX=CONST$INDX-CONST$PN$SIZE(CONST$PN$PTR);
 END ENTR$CONS$ID;
  /*****************************************************
   * ENTER$CONSTANT$ENTRY - THIS PROCEDURE *
   * DETERMINES WHICH TYPE OF CONSTANT ENTRY IS *
   * TO BE ENTERED IN THE SYMBOL TABLE, AND *
   * AND CALLS THE CORRESPONDING PROCEDURE TO *
   * MAKE THE ENTRY. *
   *****************************************************/
ENTR$CONS$NTRY: PROC PUB;
    VECPTR=VECPTR-1;
    DO CASE EXPRESS$STK(SP);
       /* CASE CONSTANT NUMBER */
       CALL ENTR$CONS$NUM;
       /* CASE IDENTIFIER CONSTANT */
       CALL ENTR$CONS$ID(POS,SP);
       /* CASE SIGNED IDENTIFIER CONSTANT */
       CALL ENTR$CONS$ID(NEG,SP);
       /* CASE CONSTANT STRING */
       DO;
         CALL SETADDRPTR(4); BYTEPTR(0)=18H OR CONS$ENTRY;
         CALL ENTER$STRING(SP);
         CONST$PN$PTR=CONST$PN$PTR-1;
         CONST$INDX=CONST$INDX-CONST$PN$SIZE(CONST$PN$PTR);
       END;
    END; /* OF CASE CONST$TYPE */
 END ENTR$CONS$NTRY;
```

```
/*******************************************************
/* ENTR$CPLX$TYP - THIS PROCEDURE IS */
/* CALLED TO ENTER THE "LINKS" AND "FORM" FOR */
/* THE 'COMPLEX TYPE' SYMBOL TABLE ENTRIES. */
/* NOTE* THAT THIS ENTRY NEVER HAS A PRINT- */
/* NAME ASSIGNED. */
/*******************************************************/
ENTR$CPLX$TYP: PROC(A) PUB;
    DCL A BYTE;
    CALL LIMITS(5);
    BASE,APTRADDR=SBTBL;
    ADDRPTR=0000H;
    CALL SETADDRPTR(2);
    ADDRPTR=PRV$SBT$ENTRY;
    PRV$SBT$ENTRY=BASE;
    CALL SETADDRPTR(4);
    BYTEPTR(0)=A;
    SBTBL=SBTBL+5;
 END ENTR$CPLX$TYP;
/*******************************************************/
/* ENTR$STR$TYP - THIS PROCEDURE IS */
/* CALLED BY THE 'TYPE' PRODUCTIONS: */
/* 1. SET TYPE */
/* 2. FILE TYPE */
/* 3. POINTER TYPE */
/* IT CALLS ENTR$CPLX$TYP TO SET UP ITS */
/* "LINKS" AND "FORM", THEN IT SETS A POINTER */
/* TO THE ASSOCIATED COMPLEX TYPE. */
/*******************************************************/
ENTR$STR$TYP: PROC(A) PUB;
    DCL A BYTE;
    CALL ENTR$CPLX$TYP(A);
    CALL LIMITS(2);
    CALL SETADDRPTR(5);
    ADDRPTR=TYPE$LOCT;
    SBTBL=SBTBL+2;
    TYPE$LOCT=BASE;
 END ENTR$STR$TYP;
/*******************************************************
 * ENTER$PARAMETER$TYPE - THIS PROCEDURE *
 * UTILIZES 3 BYTE OF CODE FOR EACH SUBROUT- *
 * INE PARAMETER THAT WAS RECOGNIZED AND PUTS *
 * THE FOLLOWING INFORMATION IN THE SYMBOL *
 * TABLE: 1. TYPE OF PARAMETER *
 * 2-3. RELATIVE LOCATION OF PARAMETER. *
 *******************************************************/
ENTR$PRM$TYP: PROC PUB;
    APTRADDR = PARAMNUMLOC + 1;
    ADDRPTR = SBTBL;
    SBTBL = SBTBL + 3*PARAMNUM - 3;
    BASE = LAST$SBTBL$ID;
    DO WHILE PARAMNUM <> 0;
      CALL SETADDRPTR(4);
      TEMPBYTE = BYTEPTR(0);
```

```
      APTRADDR = SBTBL;
      BYTEPTR(Ø) = TEMPBYTE;
      SBTBL = SBTBL + 1;
      CALL SET$PAST$PN(7);
      TEMPADDR = ADDRPTR;
      APTRADDR = SBTBL;
      ADDRPTR = TEMPADDR;
      SBTBL = SBTBL - 4;
      CALL SETADDRPTR(2);
      BASE = ADDRPTR;
      PARAMNUM = PARAMNUM - 1;
    END;
    APTRADDR = PARAMNUMLOC;
    SBTBL = SBTBL + 3*(BYTEPTR(Ø)+ 1);
 END ENTR$PRM$TYP;

/*************************************************************
 * PARM$BYTES - THIS PROCEDURE ENTERS THE NUMBER *
 * OF BYTES OCCUPIED BY A 'PARM' DECLARATION AS *
 * A THIRD ARGUMENT IN THE INTERMEDIATE CODE. *
 *************************************************************/

PARM$BYTES: PROC(LOC);
  DCL LOC BYTE; IF LOC=ØBH THEN
  CALL GENERATE(Ø2H); ELSE
  IF LOC=13H THEN
    CALL GENERATE(Ø8H);
  ELSE
    CALL GENERATE(Ø1H);
 END PARM$BYTES;

   /*********************************************************
   * BUILT$IN$PARAMETER - THIS PROCEDURE ENSURES *
   * A PROPER MATCH UP BETWEEN THE SUBROUTINE'S *
   * FORMAL PARAMETERS AND THE CALLING ACTUAL *
   * PARAMETERS. *
   *********************************************************/
BUILT$IN$PARM: PROC PUB;
    APTRADDR = PARMNUMLOC(SP);
    BASE = APTRADDR;
    IF BYTEPTR(Ø) = 13H THEN
    DO; /* CHECK FOR INTEGER OR REAL INPUT */
      IF NOT(((SHL((BYTEPTR(Ø) AND FORMMASK),3) OR
VAR$ENTRY)=
      (FORM$FIELD(SP) AND 7FH))
      OR ((ROR((BYTEPTR(Ø) AND 7ØH),1) OR VAR$ENTRY)=
      (FORM$FIELD(SP) AND 7FH))) THEN
        CALL ERROR('IP');
      ELSE
      DO;
       CALL GEN$ADDR(PARM,PRT$ADDR(SP));
       CALL PARM$BYTES(BYTEPTR(Ø));
      END;
    END;
```

```
    ELSE DO;
      IF BYTEPTR(Ø) = ØF3H THEN
      DO;
        IF SHR(FORM$FIELD(SP),3) = Ø3H THEN /* CAN'T BE */
          CALL ERROR('IP');
         ELSE
          DO;
          CALL GEN$ADDR(PARM,PRT$ADDR(SP));
          CALL PARM$BYTES(BYTEPTR(Ø));
          END;
      END;
      ELSE DO;
        IF NOT((SHL((BYTEPTR(Ø) AND FORMMASK),3) OR
VAR$ENTRY) =
         FORM$FIELD(SP)) THEN
          CALL ERROR('IP');
         ELSE
          DO;
          CALL GEN$ADDR(PARM,PRT$ADDR(SP));
          CALL PARM$BYTES(BYTEPTR(Ø));
          END;
      END;
    END;
    PARMNUMLOC(SP+2) = PARMNUMLOC(SP) + 1;
    IF SHR(FORM$FIELD(SP),7) THEN CALL GENERATE(LODI);
 END BUILT$IN$PARM;
  /****************************************************
   * ASSIGN$PARAMETERS - THIS PROCEDURE ENSURES *
   * A PROPER MATCH UP BETWEEN THE SUBROUTINE'S *
   * FORMAL PARAMETERS AND THE CALLING ACTUAL *
   * PARAMETERS. *
   ****************************************************/
ASSIGN$PARMS: PROC PUB;
    IF SIGN$FLAG THEN
    DO;
      IF FORM$FIELD(MP-3) = BUILT$IN$FUNC THEN
        CALL BUILT$IN$PARM;
    END;
    ELSE IF FORM$FIELD(MP-2) = BUILT$IN$FUNC THEN
      CALL BUILT$IN$PARM;
    ELSE DO;
      APTRADDR = PARMNUMLOC(SP);
      BASE = APTRADDR;
      IF SHR(BYTEPTR(Ø),7) THEN
      DO;
        IF (BYTEPTR(Ø) AND 7FH) = FORM$FIELD(SP) THEN
          /* THIS IS A VARIABLE PARAMETER */
          CALL GEN$ADDR(PARMV,PRT$ADDR(SP));
        ELSE CALL ERROR('IP');
      END;
      ELSE DO; /* THIS IS A VALUE PARAMETER */
        IF (BYTEPTR(Ø) = FORM$FIELD(SP))
        OR (BYTEPTR(Ø) = (FORM$FIELD(SP) AND 7FH)) THEN
          DO;
```

```
          CALL GEN$ADDR(PARM,PRT$ADDR(SP));
          CALL PARM$BYTES(BYTEPTR(0));
          END;
        ELSE CALL ERROR('IP');
      END;
      PARMNUMLOC(SP+2) = PARMNUMLOC(SP) + 3;
      READ$PARMS = TRUE;
    END;
 END ASSIGN$PARMS;
  /*****************************************************/
  /* LOOKUP$IDENTIFIER - THIS PROCEDURE IS CALLED*/
  /* WITH 'SYMHASH' AND PRINTNAME SET. IT WILL */
  /* RETURN TRUE IF THE IDENTIFIER CAN BE FOUND */
  /*****************************************************/
LOOKUP$IDENT: PROC BYTE PUB;
    BASE=HASHTABLE(SYMHASH);
    DO WHILE (BASE <> 0) AND (SBTBL > SCOPE(SCOPE$NUM));
      IF CHK$PRT$NAME(6) THEN
      DO;
        LOOKUP$ADDR=BASE;
        RETURN TRUE;
      END;
      ELSE DO;
        CALL SETADDRPTR(0);
        BASE=ADDRPTR;
      END;
    END;
    RETURN FALSE;
 END LOOKUP$IDENT;
  /*****************************************************/
  /* LOOKUP$PRINTNAME$ONLY - THIS PROCEDURE SETS */
  /* THE "SYMHASH" AND CALLS LOOKUP$IDENT TO */
  /* DETERMINE IF THE ENTRY IS IN THE SYMBOL */
  /* TABLE. THE ADDRESS OF THE PRINTNAME IS */
  /* PASSED AS A PARAMETER. IF THE ENTRY IS */
  /* FOUND, TRUE IS RETURNED. */
  /*****************************************************/
LOOKUP$PNAME: PROC(A) BYTE PUB;
    DCL A ADDR; /* ADDR OF PRINT-NAME */
    DCL B BYTE,(N BASED A)(1) BYTE;
    HASHCODE=0;
    DO B=1 TO N(0);
      HASHCODE=(HASHCODE+N(B)) AND HASHMASK;
    END;
    SYMHASH=HASHCODE;
    PRINTNAME=A;
    RETURN LOOKUP$IDENT;
 END LOOKUP$PNAME;
 /*****************************************************/
 /* STORE$CONSTANT IDENTIFIER - THIS ROUTINE IS */
 /* CALLED WITH PRINTNAME SET TO LOAD AN */
 /* IDENTIFIER IN THE 'CONSTANT VALUE' VARIABLE.*/
 /*****************************************************/
STORE$CONST: PROC PUB;
```

```
    DCL (N BASED PRINTNAME)(1) BYTE;
    CALL SETLOOKUP(SP);
    CALL MOVE(PRINTNAME,.CONST$VALUE(CONST$INDX),(N(0)+1));
    CONST$INDX=CONST$INDX+(N(0)+1);
    CONST$PN$BASE(CONST$PN$PTR)=SYMHASH;
    CONST$PN$SIZE(CONST$PN$PTR)=N(0)+1;
    CONST$PN$PTR=CONST$PN$PTR+1;
END STORE$CONST;
END SYMBOL;
```

```
$PAGEWIDTH(80) TITLE('SYNTH1 - PRODUCTION PROCEDURES')
SYNTH1: DO;

DECLARE LIT LITERALLY 'LITERALLY',
        DCL LIT 'DECLARE',
        EXT LIT 'EXTERNAL',
        POS LIT '0',
        NEG LIT '1',
        PROC LIT 'PROCEDURE',
        TRUE LIT '1',
        ADDR LIT 'ADDRESS',
        FALSE LIT '0',
        STATESIZE LIT 'ADDRESS',
        BUILT$IN$FUNC LIT '0DH'; DCL
        PSTACKSIZE LIT '48', /* SIZE OF PARSE STACKS */
        HASHTBLSIZE LIT '128', /* SIZE OF HASHTABLE */
        BCDSIZE LIT '8', /* BYTES USED FOR BCD VALUES */
        MAX$NEST LIT '3',/* MAX LEVEL OF NESTS FOR TYPES */
        MAX$ARRY$DIM LIT '5', /* MAX ARRY DIMENSIONS */
        FORMMASK LIT '7',/* USED TO DETERMINE FORM TYPE */

/* FORM ENTRIES */
        LABL$ENTRY LIT '0',
        CONS$ENTRY LIT '1',
        TYPE$ENTRY LIT '2',
        VAR$ENTRY LIT '3',
        FUNC$ENTRY LIT '5',
        TYPE$DCLE LIT '7',

/* NUMBER TYPES */
        ORD$TYPE LIT '0',
        INTEGER$TYPE LIT '1',
        CHAR$TYPE LIT '2',
        UNSIGN$EXPON LIT '3',
        SIGNED$EXPON LIT '4',
        BOOLEAN$TYPE LIT '5',
        REAL$TYPE LIT '2',
        COMPLEX$TYPE LIT '4',
        STRING$TYPE LIT '4'; $EJECT
        /* MANY OF THE FOLLOWING VARIABLES CAN BE REPLACED
BY
           MAKING USE OF THE PARALLEL PARSE STACKS */ DCL
ARRY$DIM$LOWVAL(25) ADDR EXT, ARRY$DIM$HIVAL(25) ADDR EXT,
DISP$VEC(25) ADDR EXT, ARRY$OFFSET ADDR EXT,
        CONST$PTR BYTE EXT,
        CONST$TYPE BYTE EXT,
        VECPTR BYTE EXT,
        TYPENUM BYTE EXT,
        STARTBDOS ADDR EXT, /*ADDR OF PTR TO TOP OF BDOS*/
        MAX BASED STARTBDOS ADDR ,
        TYPE$LOCT ADDR EXT,
```

```
        VAR$PTR BYTE EXT,
        ALOCBASICTYP BYTE EXT,
        ARRY$QTY(MAX$ARRY$DIM) ADDR EXT,
        VAR$BASE(10) ADDR EXT,
        VAR$BASE1(10) ADDR EXT,
        ALLC$QTY ADDR EXT,
        PARENT$TYPE ADDR EXT,
        CONST$INDX BYTE EXT,
        LOOKUP$ADDR ADDR EXT,
        CONST$VALUE(16) BYTE EXT,
        CONST$PN$BASE(4) BYTE EXT,
        CONST$PN$PTR BYTE EXT,
        CONST$PN$SIZE(4) BYTE EXT,
        INTEGER$DIFF ADDR EXT,
        SUBR$VAL(2) ADDR EXT,
        SUBR$TYPE(2) BYTE EXT,
        SUBR$PTR BYTE EXT,
        SUB$TYP$ADDR(1) ADDR EXT,
        SUBR$FORM BYTE EXT,
        SIGNVALU BYTE EXT,
        ARRY$BASE ADDR EXT,
        ARRY$PTR BYTE EXT,
        ARRY$DIM$PTR BYTE EXT,
        PTRPTR BYTE EXT,
        REC$VAR$TYP(MAX$NEST) BYTE EXT,
        REC$NST BYTE,
        VARIANT$PART(MAX$NEST) BYTE EXT,
        NUM$ARRY$DIM(MAX$ARRY$DIM) BYTE EXT,
        ARRY$DIMEN(25) ADDR EXT,
        CONST$NUM$TYPE(4) BYTE EXT,
        ARY$DM$ADR$PTR BYTE EXT;

DCL BCDNUM(BCDSIZE) BYTE EXT,
        SCOPE(10) ADDR EXT,
        SCOPE$NUM BYTE EXT,
        TEMPBYTE BYTE EXT,
        TEMPBYTE1 BYTE EXT,
        TEMPADDR ADDR EXT,
        TEMPADDR1 ADDR EXT;

DCL
              /* COUNTERS */
         CODESIZE ADDR EXT, /* COUNTS NUMBER OF LABELS */
         ERRORCOUNT ADDR EXT, /* COUNTS NUMBER OF ERRORS */
         ALLOC$ADDR ADDR EXT, /* COUNTS PRT ENTRIES */
              /* FLAGS USED DURING CODE GENERATION */
         WRITE$STMT BYTE EXT, /* IN WRITE STATEMENT */
         READ$STMT BYTE EXT, /* IN READ STATEMENT */
         NEW$STMT BYTE EXT, /* GETS NEW RECORD */
         DISPOSE$STMT BYTE EXT, /* DISPOSES OF RECORD */
         ALLOCATE BYTE EXT, /* PRT LOCATION ASSIGNED */
         VARPARM BYTE EXT, /* FORMAL PARAM IS VARIABLE TYPE
*/
         READPARMS BYTE EXT, /* READING ACTUAL PARAMETERS */
```

```
        PRESENT BYTE EXT, /* IDENTIFIER IS IN SYMBOL TABLE
*/

         /* GLOBAL VARIABLES USED BY THE SCANNER */
        TOKEN BYTE EXT, /* TYPE OF TOKEN JUST SCANNED */
         /* GLOBAL VARIABLES USED IN SYMBOL TABLE
OPERATIONS */
        BASE ADDR EXT, /* BASE LOCATION OF ENTRY */
        HASHTABLE(HASHTBLSIZE) ADDR EXT, /* HASHTABLE ARRAY
*/
        SBTBLTOP ADDR EXT, /* HIGHEST LOCATION OF SYMBOL
TABLE */
        SBTBL ADDR EXT, /* CURRENT TOP OF SYMBOL TABLE */
        APTRADDR ADDR EXT, /* UTILITY VARIABLE TO ACCESS
SBTBL */
        ADDRPTR BASED APTRADDR ADDR, /* CURRENT 2 BYTES
POINTED AT */
        (BYTEPTR BASED APTRADDR)(1) BYTE, /* CURRENT BYTE
POINTED AT */
        PRINTNAME ADDR EXT, /* SET PRIOR TO LOOKUP OR ENTER
*/
        SYMHASH BYTE EXT, /* HASH VALUE OF AN IDENTIFIER */
        LAST$SBTBL$ID ADDR EXT, /* HOLD PREVIOUS BASE
LOCATION */
        PARAMNUMLOC ADDR EXT, /* STORES POINTER TO PARAM
LISTING */
        SBTBLSCOPE ADDR EXT; /* BASE OF LAST ENTRY IN
PREVIOUS BLOCK*/

DCL BUILT$IN$TBL(12) BYTE EXT;

/*********PARSER VARIABLES*********/

DCL PARMNUM(PSTACKSIZE) BYTE EXT, /* MAINTAINS NUMBER OF
PARAMETERS ASSOCIATED WITH A SUBROUTINE */
       LABELSTACK(PSTACKSIZE) ADDR EXT, /* TRACKS STATEMENT
LABELS */
       PARMNUMLOC(PSTACKSIZE) ADDR EXT, /* MAINTAINS THE
LOCATION IN SYMBOL TBL WHERE PARAMETER INFO STORED */
       BASE$LOC(PSTACKSIZE) ADDR EXT, /* STORES THE SYMBOL
TABLE ADDRESS OF THE PERTINATE ENTRY */
       FORM$FIELD(PSTACKSIZE) BYTE EXT, /* STORES THE FORM
FIELD OF SCANNED IDENTIFIERS */
       TYPE$STACK(PSTACKSIZE)BYTE EXT,/* HOLDS A VARIABLE´S
TYPE */
       EXPRESS$STK(PSTACKSIZE)BYTE EXT, /* CONTAINS THE
TYPES OF THE EXPRESSION COMPONENTS */
       PRT$ADDR(PSTACKSIZE) ADDR EXT, /* STORES AN
IDENTIFIER´S PRT LOCATION */
       PARAMNUM BYTE EXT,
       (SP,MP,MPP1) BYTE EXT;

$EJECT
    /* MNEMONICS FOR PASCAL-SM MACHINE */
```

```
 DCL NOP LIT '0',ENDP LIT '1',LBL LIT '2',LDIB LIT '3',
      LDII LIT '4',PRO LIT '5',RTN LIT '6',SAVP LIT '7',
      UNSP LIT '8',CNVB LIT '9',CNVI LIT'10',ALL LIT'11',
      LITA LIT'12',ADDB LIT'13',ADDI LIT'14',SUBB LIT'15',
      SUBI LIT'16',MULB LIT'17',MULI LIT'18',DIVB LIT'19',
      DIVI LIT'20',MODX LIT'21',EQLI LIT'22',NEQI LIT'23',
      LEQI LIT'24',GEQI LIT'25',LSSI LIT'26',GRTI LIT'27',
      XIN LIT'28',EQLB LIT'29',NEQB LIT'30',LEQB LIT'31',
      GEQB LIT'32',LSSB LIT'33',GRTB LIT'34',EQLS LIT'35',
      NEQS LIT'36',LEQS LIT'37',GEQS LIT'38',LSSS LIT'39',
      GRTS LIT'40',EQSET LIT'41',NEQST LIT'42',INCL1
LIT'43',
      INCL2 LIT'44',NEGB LIT'45',
      NEGI LIT'46',COMB LIT'47',COMI LIT'48',NOTX LIT'49',
      ANDX LIT'50',BOR LIT'51',STOB LIT'52',STOI LIT'53',
      STO LIT'54',STDB LIT'55',STDI LIT'56',STD LIT'57',
      UNION LIT'58',STDIF LIT'59',ISEC LIT'60',CNAI LIT'61',
      BRL LIT'62',BLC LIT'63',CN2I LIT'64',MKSET LIT'65',
      XCHG LIT'66',PARM LIT'67',PARMV LIT'68',PARMX LIT'69',
      INC LIT'70',DEC LIT'71',DEL LIT'72',WRT LIT'73',
      SUB LIT'74',LDSI LIT'75',KASE LIT'76',LOD LIT'77',
      LODB LIT'78',LODI LIT'79',RDVB LIT'80',RDVI LIT'81',
      RDVS LIT'82',WRTB LIT'83',WRTI LIT'84',WRTS LIT'85',
      DUMP LIT'86',ABS LIT'87',SQR LIT'88',SIN LIT'89',
      COS LIT'90',ARCTN LIT'91',EXP LIT'92',LN LIT'93',
      SQRT LIT'94',ODD LIT'95',EOLN LIT'96',EOF LIT'97',
      TRUNC LIT'98',ROUND LIT'99',ORD LIT '100',CHR LIT
'101',
      SUCC LIT'102',PRED LIT'103',SEEK LIT'104',PUT
LIT'105',
      GET LIT'106',RESET LIT'107',REWRT LIT'108',PAGE
LIT'109',
      NEW LIT'110',DISPZ LIT'111',FWD LIT'112',XTRNL
LIT'113',
      RDV LIT'114';

$EJECT ERROR:PROC(ERRCODE) EXTERNAL;
  DCL ERRCODE ADDR;
 END ERROR;

LOOKUP$ONLY:PROC (A) BYTE EXTERNAL;
  DCL A BYTE;
 END LOOKUP$ONLY;

MOVE: PROC (SOURCE,DESTIN,L) EXTERNAL;
  DCL (SOURCE,DESTIN) ADDR,
      L BYTE;
 END MOVE;

SETADDRPTR: PROC(OFFSET) EXTERNAL;
   DCL OFFSET BYTE;
 END SETADDRPTR;

MON3:PROC EXTERNAL;
```

```
 END MONS;

LIMITS:PROC(COUNT) EXTERNAL;
   DCL COUNT BYTE;
 END LIMITS;

ENTR$CPLX$TYP: PROC (A) EXTERNAL;
   DCL A BYTE;
 END ENTR$CPLX$TYP;

SET$PAST$PN:PROC(OFFSET) EXTERNAL;
   DCL OFFSET BYTE;
 END SET$PAST$PN;

LOOKUP$PNAME:PROC(A) BYTE EXTERNAL;
   DCL A ADDR;
 END LOOKUP$PNAME;

GENERATE:PROC(OBJCODE) EXTERNAL;
   DCL OBJCODE BYTE;
 END GENERATE;

GEN$ADDR:PROC(A,B) EXTERNAL;
   DCL A BYTE, B ADDR;
 END GEN$ADDR;

ASSIGN$PARMS:PROC EXTERNAL;
 END ASSIGN$PARMS;

ENTER$VAR$ID:PROC (A,B,ID$ENTRY) EXTERNAL;
   DCL (A,B,ID$ENTRY) BYTE;
 END ENTER$VAR$ID;

ENTER$LABEL:PROC EXTERNAL;
 END ENTER$LABEL;

ENTR$PRM$TYP:PROC EXTERNAL;
 END ENTR$PRM$TYP;

PRINTCHAR:PROC (CHAR) EXTERNAL;
    DCL CHAR BYTE;
 END PRINTCHAR;

CRLF:PROC EXTERNAL;
 END CRLF;

PRINT$ERROR:PROC EXTERNAL;
 END PRINT$ERROR;

WRIT$INT$FILE:PROC EXTERNAL;
 END WRIT$INT$FILE;

MOVE$SBTBL:PROC EXTERNAL;
 END MOVE$SBTBL;
```

```
CLOSE$INT$FIL:PROC EXTERNAL;
 END CLOSE$INT$FIL;

PRINT:PROC(A) EXTERNAL;
  DCL A ADDR;
 END PRINT;

LOOKUP$IDENT:PROC BYTE EXTERNAL;
 END LOOKUP$IDENT;

$EJECT

INIT$SYNTH: PROC PUBLIC;
   CODESIZE = 0;
   SBTBLTOP=MAX-2;
   VECPTR=0;
   CONST$PTR=0;
   CONST$INDX=0;
   CONST$PN$PTR=0;
   SUBR$PTR=0;
   ARY$DM$ADR$PTR=-1;
   ARRY$PTR=-1;
   VARIANT$PART(0)=FALSE;
   ARRY$QTY(0)=0;
   ALLOC$ADDR=0;
 END INIT$SYNTH;


   /*********************************************************
    * SUBRANGE$ERROR - THIS PROCEDURE IS CALLED *
    * IN THE EVENT OF AN IMPROPER VALUE IN A *
    * SUBRANGE. *
    *********************************************************/
SUBR$ERROR: PROC;
    CALL ERROR('IS');
    SUBR$TYPE(SUBR$PTR)=INTEGER$TYPE;
    SUBR$VAL(SUBR$PTR)=0000E;
 END SUBR$ERROR;


   /*********************************************************
    * ORD$HIGH$LOW$CHECK - THIS PROCEDURE IS *
    * CALLED TO ENSURE THE SECOND SUBRANGE VALUE *
    * IS GREATER THAN THE FIRST. *
    *********************************************************/
ORD$HI$LOW$CHK: PROC PUBLIC;
    IF SUBR$PTR=0 THEN RETURN;
    IF SUBR$TYPE(0)=SUBR$TYPE(1) THEN
      IF SUBR$VAL(0) > SUBR$VAL(1) THEN RETURN;
    CALL ERROR('IS');
 END ORD$HI$LOW$CHK;
```

```
/*********************************************************
 * SUBRANGE$INTEGER$HI$LO$CHECK - THIS PROCE- *
 * DURE IS CALLED TO ENSURE THAT BOTH SUB- *
 * RANGE ELEMENTS ARE OF THE SAME TYPE, AND *
 * THAT THEIR VALUES DO NOT EXCEED THE MAX *
 * INTEGER VALUE. *
 *********************************************************/
SUB$INT$HL$CHK: PROC;
   IF SUBR$PTR=0 THEN RETURN;
   IF SUBR$TYPE(0) <> SUBR$TYPE(1) THEN
   DO;
     CALL SUBR$ERROR;
     RETURN;
   END;
   IF SUBR$VAL(0) < 32768 AND SUBR$VAL(1) >32767 THEN
   DO;
     INTEGER$DIFF = SUBR$VAL(0)+( -SUBR$VAL(1))+1;
     RETURN;
   END;
   IF SUBR$VAL(0) > 32767 AND SUBR$VAL(1) < 32768 THEN
   DO;
     CALL SUBR$ERROR;
     RETURN;
   END;
   IF SUBR$VAL(0) < 32768 THEN /* BOTH POSITIVE */
   DO;
     IF(SUBR$VAL(0)-(SUBR$VAL(1)+1)) < 32768 THEN
     DO;
       INTEGER$DIFF=SUBR$VAL(0)-(SUBR$VAL(1))+1;
       RETURN;
     END;
     CALL SUBR$ERROR;
     RETURN;
   END;
   ELSE /* BOTH NEGATIVE */
     IF ( - SUBR$VAL(1)-( - SUBR$VAL(0) +1)) < 32768 THEN
     DO;
       INTEGER$DIFF=( - SUBR$VAL(1))-( - SUBR$VAL(0))+1;
       RETURN;
     END;
   CALL SUBR$ERROR;
 END SUB$INT$HL$CHK;


/*********************************************************/
/* SUBRANGE$IDENTIFER$PROCEDURE - THIS ROUTINE */
/* IS CALLED TO DETERMINE THE OFFSET ( NUMBER */
/* OF ENTRIES IN A SUBRANGE ) AND THE TYPE OF */
/* SUBRANGE, GIVEN THAT THE SUBRANGE TYPE IS */
/* A NAMED IDENTIFIER. */
```

```
/*****************************************************************/
SUB$ID$PROC: PROC;
    CONST$PN$PTR=CONST$PN$PTR-1;
    CONST$INDX=CONST$INDX-CONST$PN$SIZE(CONST$PN$PTR);
    PRINTNAME=.CONST$VALUE(CONST$INDX);
    SYMHASH=CONST$PN$HASH(CONST$PN$PTR);
    IF NOT LOOKUP$IDENT THEN CALL SUBR$ERROR;
    ELSE DO; /* FOUND CONSTANT IDENTIFIER */
      BASE=LOOKUP$ADDR;
      CALL SETADDRPTR(4); /* POINTS TO FORM(BYTEPTR) */
      SUBR$FORM=BYTEPTR(0);
      IF SUBR$FORM <> 07H AND (SUBR$FORM AND FORMMASK) <>
CONS$ENTRY
        THEN CALL SUBR$ERROR;
      ELSE DO;
        IF SUBR$FORM = 07H THEN
        DO;
          SUBR$TYPE(SUBR$PTR)=ORD$TYPE;
          CALL SETADDRPTR(6);
          SUBR$FORM=BYTEPTR(0); /* LENGTH OF P.NAME */
          CALL SETADDRPTR(7+SUBR$FORM);
          SUBR$VAL(SUBR$PTR)=DOUBLE(BYTEPTR(0));
          CALL SETADDRPTR(7+SUBR$FORM);
          SUB$TYP$ADDR(SUBR$PTR)=ADDRPTR;
          CALL ORD$FI$LOW$CHK;
        END;
        ELSE DO;
          DO WHILE ((SHR(SUBR$FORM,3) AND 3H)=0);
            IF SHR(SUBR$FORM,5)=NEG THEN
              IF SIGNVALU=POS THEN SIGNVALU=NEG;
              ELSE SIGNVALU=POS;
            CALL SETADDRPTR(6);
            SUBR$FORM=BYTEPTR(0);
            CALL SETADDRPTR(7+SUBR$FORM);
            IF NOT LOOKUP$ONLY(APTRADDR) THEN
            DO;
              CALL SUBR$ERROR;
              SUBR$PTR=SUBR$PTR+1;
              RETURN;
            END;
            ELSE DO;
              BASE=LOOKUP$ADDR;
              CALL SETADDRPTR(4);
              SUBR$FORM=BYTEPTR(0);
            END;
          END;
          IF (SHR(SUBR$FORM,3)AND 3H) = 2 THEN
          DO;
            CALL SUBR$ERROR;
            SUBR$PTR=SUBR$PTR+1;
            RETURN;
          END;
          /* HERE WE HAVE EITHER AN INTEGER OR CHAR */
          IF (SHR(SUBR$FORM,3) AND 3H) = 1 THEN
```

```
      DO; /* INTEGER */
        CALL SETADDRPTR(6);
        SUBR$FORM=BYTEPTR(0);
        CALL SETADDRPTR(7+SUBR$FORM);
        IF SIGNVALU = NEG THEN
          SUBR$VAL(SUBR$PTR)= - ADDRPTR;
        ELSE SUBR$VAL(SUBR$PTR)=ADDRPTR;
        SUBR$TYPE(SUBR$PTR)=INTEGER$TYPE;
        CALL SUB$INT$HL$CHK;
      END;
      ELSE
      DO;
        CALL SETADDRPTR(6);
        SUBR$FORM=BYTEPTR(0);
        CALL SETADDRPTR(7+SUBR$FORM);
        IF BYTEPTR(0) <> 1 THEN
        DO;
          CALL SUBR$ERROR;
          SUBR$PTR=SUBR$PTR+1;
          RETURN;
        END;
        CALL SETADDRPTR(8+SUBR$FORM);
        IF BYTEPTR(0) <41H OR BYTEPTR(0) > 5AH THEN
          CALL SUBR$ERROR;
        ELSE DO;
          SUBR$VAL(SUBR$PTR)=DOUBLE(BYTEPTR(0)-41H);
          SUBR$TYPE(SUBR$PTR)=CHAR$TYPE;
          CALL ORD$HI$LOW$CHK;
        END;
      END;
    END;
  END;
 END;
 SUBR$PTR=SUBR$PTR+1;
END SUB$ID$PROC ;


/*****************************************************/
/* SUBRANGE$CASE - THIS PROCEDURE IS USED TO */
/* DETERMINE THE NUMBER OF ENTRIES IN A SUBRANGE*/
/*****************************************************/
SUBR$CASE: PROC(A);
  DCL A BYTE;
  SIGNVALU=POS;
  DO CASE EXPRESS$STK(A);
   /* CASE CONST NUMBER */
   DO; CONST$PTR=CONST$PTR-1;
     IF CONST$NUM$TYPE(CONST$PTR)=REAL$TYPE THEN
       DO;
         CALL SUBR$ERROR;
         CONST$INDX=CONST$INDX-BCDSIZE;
       END;
     ELSE
```

```
        DO; /* INTEGER TYPE */
          CONST$INDX=CONST$INDX-2;
          CALL
MOVE(.CONST$VALUE(CONST$INDX),.SUBR$VAL(SUBR$PTR),2);
          SUBR$TYPE(SUBR$PTR)=INTEGER$TYPE;
          CALL SUB$INT$FL$CHK;
        END;
      SUBR$PTR=SUBR$PTR+1; /* NEXT TO FILL */
     END;
     /* CASE IDENT CONSTANT */
     CALL SUB$ID$PROC;
     /* CASE SIGNED IDENT CONSTANT */
     DO;
       SIGNVALU=NEG;
       CALL SUB$ID$PROC;
     END;
     /* CASE CONSTANT STRING */
     DO;
        CONST$PN$PTR=CONST$PN$PTR-1;
       CONST$INDX=CONST$INDX-CONST$PN$SIZE(CONST$PN$PTR);
       PRINTNAME=.CONST$VALUE(CONST$INDX);
       IF CONST$PN$SIZE(CONST$PN$PTR) <> 2 THEN
         CALL SUBR$ERROR;
       ELSE
         DO;
           BASE=PRINTNAME;
           CALL SETADDRPTR(1);
           IF BYTEPTR(0) < 41H OR BYTEPTR(0) > 5AH THEN
             CALL SUBR$ERROR;
           ELSE
             DO;
               SUBR$VAL(SUBR$PTR)=DOUBLE(BYTEPTR(0)-41H);
               SUBR$TYPE(SUBR$PTR)=CHAR$TYPE;
               CALL ORD$HI$LOW$CHK;
             END;
         END;
       SUBR$PTR=SUBR$PTR+1;
     END;
   END; /* OF CASE EXPRESS$STK(MP) */
 END SUBR$CASE;


 /*****************************************************/
 /* ENTER$SUBRANGE$ENTRY - THIS PROCEDURE IS */
 /* USED TO ENTER A SUBRANGE TYPE ENTRY INTO */
 /* THE SYMBOL TABLE. THIS SYMBOL TABLE ENTRY */
 /* HAS NO PRINTNAME ASSOCIATED WITH IT. */
 /*****************************************************/
ENTR$SUB$NTRY: PROC PUBLIC;
    TYPE$LOCT=SBTBL;
    CALL LIMITS(12);
    VECPTR=VECPTR-1;
    CALL SUBR$CASE(SP);
```

```
    VECPTR=VECPTR-1;
    CALL SUBR$CASE(MP);
    CALL ENTR$CPLX$TYP(SHL(SUBR$TYPE(0),6)OR 0FEH);
    CALL SETADDRPTR(5);
    IF SUBR$TYPE(0)=INTEGER$TYPE THEN
      ADDRPTR=.BUILT$IN$TBL;
    IF SUBR$TYPE(0)=CHAR$TYPE THEN
ADDRPTR=(.BUILT$IN$TBL+23);
    IF SUBR$TYPE(0)=ORD$TYPE THEN ADDRPTR=SUB$TYP$ADDR(0);
    CALL SETADDRPTR(7);
    ADDRPTR=SUBR$VAL(1);
    CALL SETADDRPTR(9);
    ADDRPTR=SUBR$VAL(0);
    CALL SETADDRPTR(11);
    IF SUBR$TYPE(0)=INTEGER$TYPE THEN /* RANGE 0 TO 64K */
      ADDRPTR=INTEGER$DIFF; /* MAY BE GREATER THAN 32767 */
    ELSE
      ADDRPTR=((SUBR$VAL(0)-SUBR$VAL(1))+1);
    SUBR$PTR=0;
    SBTBL=SBTBL+8;
 END ENTR$SUB$NTRY;



   /*****************************************************
   * TYPE$ERROR - THIS PROCEDURE IS CALLED IN THE*
   * EVENT OF AN INCOMPATIBLE TYPE. *
   *****************************************************/
TYPE$ERROR: PROC;
    ALLOCATE=FALSE;
    CALL ERROR('IT');
 END TYPE$ERROR;



 /******************************************************/
 /* ALLOCATE OFFSET - THIS PROCEDURE IS CALLED TO*/
 /* DETERMINE THE NUMBER OF BYTES REQUIRED FOR */
 /* STORAGE OF A VARIABLE OF THE TYPE GIVEN IN */
 /* THE PARAMETER 'A'. THE VARIABLE'S ALLC$QTY */
 /* AND ALLC$FORM ARE SET UPON RETURN. */
 /******************************************************/
ALLC$OFFSET: PROC(A) PUBLIC; /* TYPE$LOCT */
    DCL A ADDR;
    DCL (ALLC$FORM,B) BYTE;
    BASE=A;
    CALL SETADDRPTR(4); /* POINTS TO FORM OF TYPE */
    ALLC$FORM= BYTEPTR(0) AND FORMMASK;
    IF ALLC$FORM <> TYPE$ENTRY AND ALLC$FORM <> TYPEDCLE
THEN
      DO;
      CALL TYPE$ERROR;
      ALLC$QTY=1;
      ALOCBASICTYP=0;
```

```
      RETURN;
    END;
    DO WHILE((SHR(BYTEPTR(0),3)AND FORMMASK)=7 AND
ALLC$FORM=TYPE$ENTRY);
      CALL SET$PAST$PN(7);
      BASE=ADDRPTR; CALL SETADDRPTR(4);
      TYPE$LOCT = BASE;
      ALLC$FORM=BYTEPTR(0) AND FORMMASK;
      IF ALLC$FORM <> TYPE$ENTRY AND ALLC$FORM <> TYPEDCLE
THEN
       DO: CALL TYPE$ERROR;
         ALLC$QTY=1;
         ALOCBASICTYP=0; RETURN;
       END;
    END;
 /* HERE EXISTS EITHER A BASIC TYPE OR A TYPE DECLARATION */
    IF ALLC$FORM = TYPE$ENTRY THEN
     DO; /* BASIC TYPE */
      DO CASE (SHR(BYTEPTR(0),3) AND FORMMASK);
        /* INTEGER */
       DO:
         ALLC$QTY=2;
         ALOCBASICTYP=INTEGER$TYPE;
       END;
        /* BCD REAL */
       DO:
         ALLC$QTY=8;
         ALOCBASICTYP=UNSIGN$EXPON;
       END;
        /* CHARACTER */
       DO;
         ALLC$QTY=1;
         ALOCBASICTYP=CHAR$TYPE;
       END;
        /* BOOLEAN */
       DO;
         ALLC$QTY=1;
         ALOCBASICTYP=BOOLEAN$TYPE;
       END;
        /* TEXT */
       DO:
         ALLC$QTY = 2;
         ALOCBASICTYP = STRING$TYPE;
       END;
     END; /* OF CASE */
     ALLOCATE=TRUE;
     RETURN;
   END; /* HERE EXISTS A TYPE DECLARATION */
   TEMPBYTE1,ALLC$FORM=(SHR(BYTEPTR(0),3)AND FORMMASK);
   IF ALLC$FORM=0 THEN
    DO; /* SCALAR */
     ALLOCATE=TRUE;
     ALLC$QTY=DOUBLE(ALLC$FORM+1);
     ALOCBASICTYP=ORD$TYPE; RETURN;
```

```
    END;
   IF ALLC$FORM=1 THEN
    DO; /* SUBRANGE */
     ALLOCATE=TRUE;
     ALOCBASICTYP=COMPLEX$TYPE;
     B=SHR(BYTEPTR(0),6);
     IF B = 1 THEN ALLC$QTY=DOUBLE(ALLC$FORM+1);
     ELSE ALLC$QTY=DOUBLE(ALLC$FORM); RETURN;
    END;
   IF ALLC$FORM=2 THEN
    DO; /* ARRAY */
     ALLOCATE=TRUE;
     ALOCBASICTYP=COMPLEX$TYPE;
     CALL SETADDRPTR(8);
     ALLC$QTY=ADDRPTR; RETURN;
    END;
   B=2;
   /* ALL OTHER CASES ALLOCATE AN ADDRESS FIELD */
   ALLC$QTY=DOUBLE(B);
   ALOCBASICTYP=COMPLEX$TYPE;
   ALLOCATE=TRUE;
 END ALLC$OFFSET;



 /*************************************************************/
 /* AL$NDX$OFFSET - THIS PROCEDURE IS CALLED */
 /* TO DETERMINE THE NUMBER OF BYTES REQUIRED */
 /* BY AN ARRAY TO STORE THE ARRAY'S COMPONENTS */
 /* TYPE$LOCT IS SET PRIOR TO CALLING THIS */
 /* ROUTINE. AN ADDRESS VARIABLE CONTAINING THE */
 /* BYTE COUNT IS RETURNED. */
 /*************************************************************/
AL$NDX$OFFSET: PROC ADDR PUBLIC;
   DCL A ADDR,B BYTE;
    A,BASE=TYPE$LOCT;
    CALL SETADDRPTR(4);
    DO WHILE (SHR(BYTEPTR(0),3) AND FORMMASK) = 7 AND
             ( BYTEPTR(0) AND FORMMASK ) = TYPE$ENTRY;
     CALL SET$PAST$PN(7);
     BASE=ADDRPTR; CALL SETADDRPTR(4);
     TYPE$LOCT = BASE;
    END;
 /* HERE WE HAVE EITHER A SCALAR,SUBRANGE,BOOLEAN, OR CHAR
TYPE */
    B= SHR(BYTEPTR(0),3) AND FORMMASK;
    IF (BYTEPTR(0) AND FORMMASK) = TYPE$ENTRY THEN
     DO;
       IF B = 0 OR B = 1 THEN
         DO;
           CALL ERROR('IA');
           B=2;
           RETURN DOUBLE(B);
         END;
```

```
      IF B=2 THEN /* CHARACTER SUBRANGE */
        DO;
         B = 26;
         REC$VAR$TYP(REC$NST)=CHAR$TYPE;
         RETURN DOUBLE(B);
        END;
      /* BOOLEAN */
      REC$VAR$TYP(REC$NST)=BOOLEAN$TYPE;
      B = 2; RETURN DOUBLE(B);
     END;
 /* COMPLEX TYPE */
   IF (( BYTEPTR(0) AND FORMMASK) <> TYPE$DCLE OR
      (( B <> 0 ) AND ( B <> 1 ))) THEN
     DO;
       CALL ERROR('IA');
       B=2; RETURN DOUBLE(B);
     END;
   IF B=0 THEN
     DO; /* SCALAR TYPE */
      REC$VAR$TYP(REC$NST)=COMPLEX$TYPE;
      CALL SET$PAST$PN(7);
      RETURN DOUBLE(BYTEPTR(0) + 1);
    END;
 /* SUBRANGE TYPE */
    REC$VAR$TYP(REC$NST)=ORD$TYPE;
    CALL SETADDRPTR(11);
    RETURN ADDRPTR;
 END AL$NDX$OFFSET;



 /*************************************************
  * ALLOCATE$VARIABLES - THIS PROCEDURE IS *
  * CALLED TO ASSIGN PRT LOCATIONS FOR EACH *
  * OF THE PROGRAM VARIABLES. *
  *************************************************/
ALLOC$VARS: PROC PUBLIC;
    TEMPBYTE1 = 0;
    CALL ALLC$OFFSET(TYPE$LOCT);
    TEMPBYTE = VAR$PTR;
    DO VAR$PTR = 0 TO TEMPBYTE;
      BASE=VAR$BASE(VAR$PTR);
      CALL SETADDRPTR(4);
      IF SHR(BYTEPTR(0),7) THEN
      DO;
        BYTEPTR(0) = (BYTEPTR(0)) OR (SHL(ALOCBASICTYP,3) OR
VAR$ENTRY);
        APTRADDR = VAR$BASE1(VAR$PTR);
        ADDRPTR = ALLOC$ADDR;
        ALLOC$ADDR = ALLOC$ADDR + 2;
      END;
      ELSE DO;
        BYTEPTR(0)=SHL(ALOCBASICTYP,3) OR VAR$ENTRY;
        IF (BYTEPTR(0) = 23H) AND (TEMPBYTE1 = 2) THEN
```

```
        DO;
          APTRADDR = TYPE$LOCT + 8;
          ALLC$QTY = ADDRPTR;
        END; /* IF TEMPBYTE1 = 3 THEN
        DO;
          APTRADDR = TYPE$LOCT + 6;
          APTRADDR = APTRADDR + BYTEPTR(0) + 1;
          APTRADDR = ADDRPTR + 5;
          ALLC$QTY = ADDRPTR;
        END; */
        APTRADDR=VAR$BASE1(VAR$PTR);
        ADDRPTR=ALLOC$ADDR;
        ALLOC$ADDR=ALLOC$ADDR+ALLC$QTY;
      END;
      APTRADDR=APTRADDR+2;
      ADDRPTR=TYPE$LOCT;
    END;
    TEMPBYTE1 = 0;
 END ALLOC$VARS;


   /*****************************************************
    * CASE$PTRPTR - THIS PROCEDURE IS CALLED TO *
    * SET A VARIABLE'S APPROPRIATE TYPE. *
    *****************************************************/
CASE$PTRPTR: PROC(A) PUBLIC;
    DCL A BYTE;
    DO CASE A;
     /* CASE 0 ORD VARIABLE */
     DO;
       PTRPTR = 10H;
       CALL SET$PAST$PN(9);
       BASE$LOC(SP) = ADDRPTR; /* ADDR OF PARENT */
     END;
     /* CASE 1 INTEGER VARIABLE */
     PTRPTR = 09E;
     /* CASE 2 CHAR VARIABLE */
     PTRPTR = 3BH;
     /* CASE 3 REAL VARIABLE */
     PTRPTR = 0AE;
     /* CASE 4 COMPLEX VARIABLE */
       DO; /* ARRAY, SUBRANGE, USER DEFINED TYPES */
         TEMPADDR = BASE; /* STORE VARIABLE SBTL LOCATION */
         CALL SET$PAST$PN(9);
         BASE = ADDRPTR;
         CALL SETADDRPTR(4);
         IF BYTEPTR(0) = 17H THEN /* ARRAY */
         DO;
           APTRADDR = APTRADDR + 6;
           TEMPBYTE1 = BYTEPTR(0);
         END;
         ELSE IF (BYTEPTR(0) AND 0FE) = 0FE THEN /* SUBRANGE
TYPES */
```

```
          TEMPBYTE1 = SHR(BYTEPTR(0), 6);
        ELSE IF BYTEPTR(0) = 7AH THEN
        DO; /* USER DEFINED TYPE */
          TEMPBYTE1 = 0;
          CALL SET$PAST$PN(7);
          BASE = ADDRPTR;
          CALL SETADDRPTR(4);
          IF BYTEPTR(0) <> 27H THEN CALL ERROR('NS');
         /* THIS IS A SET TYPE */
          CALL SETADDRPTR(5);
          BASE$LOC(SP) = ADDRPTR; /* ADDR OF PARENT */
        END;
        ELSE IF BYTEPTR(0) = 37H THEN
        DO; /* POINTER */
          CALL SETADDRPTR(5);
          BASE$LOC(SP) = ADDRPTR; /* ADDR OF PARENT */
        END;
        ELSE TEMPBYTE1 = 06H;
        DO CASE TEMPBYTE1;
          PTRPTR = 10H;
          PTRPTR = 09H;
          PTRPTR = 0BH;
          PTRPTR = 0AH;
          PTRPTR = 0CH;
          PTRPTR = 08H;
          PTRPTR = 0CH;
        END; /* OF CASE */
        BASE = TEMPADDR; /* RESTORE ORIGINAL BASE LOCATION
*/
      END;
    /* CASE 5 BOOLEAN VARIABLE */
    PTRPTR = 08H;
   END; /* OF VARIABLE CASE */
 END CASE$PTRPTR;



 /****************************************************/
 /* SET$VARIABLE$TYPE - THIS PROCEDURE IS CALLED */
 /* TO SET THE VARIABLE TYPE, VARIABLE SIGN,AND */
 /* ADDRESS OF THE BASIC TYPE GIVEN. THE ADDRESS*/
 /* VARIABLE 'LOOKUP$ADDR' IS SET PRIOR TO THE */
 /* CALL. */
 /****************************************************/
SET$VAR$TYPE: PROC PUBLIC;
    SET$TYP$N$LOC: PROC(A,B,C);
    DCL (A, B, C) BYTE;
        CALL SET$PAST$PN(A);
        IF (B=04H) OR (B=05H) OR (B=06H) OR (B=11H) THEN
          PRTADDR(SP) = APTRADDR;
        ELSE PRTADDR(SP) = ADDRPTR;
        TYPE$STACK(SP) = (B OR ROL(C, 7));
     END SET$TYP$N$LOC;
    BASE = LOOKUP$ADDR;
```

```
    CALL SETADDRPTR(4);
    FORM$FIELD(SP) = BYTEPTR(0);
    DO CASE (FORM$FIELD(SP) AND FORMMASK);
      ;
      /* CONSTANT ENTRY */
      DO;
        SIGN$VALU = POS;
        DO CASE (SHR(BYTEPTR(0),3) AND 03H);
        /* FIND OUT WHAT KIND OF CONSTANT IT IS */
          DO WHILE (SHR(BYTEPTR(0),3) AND 03H) = 0;
            IF (SHR(BYTEPTR(0),5) AND 01H) = 01H THEN
              IF SIGN$VALU THEN SIGN$VALU = NEG;
              ELSE SIGN$VALU = POS;
            CALL SETADDRPTR(6);
            IF NOT LOOKUP$PNAME(APTRADDR) THEN
            DO;
              CALL ERROR('IC');
              RETURN;
            END;
            CALL SETADDRPTR(4);
            IF (BYTEPTR(0) AND FORMMASK) <> CONS$ENTRY THEN
            DO;
              CALL ERROR('IC');
              RETURN;
            END;
          END;
          /* INTEGER OR BOOLEAN CONSTANT */
          IF BASE < .MEMORY THEN /* BOOLEAN */
            CALL SET$TYP$N$LOC(9,4H,POS);
          ELSE /* INTEGER */
            CALL SET$TYP$N$LOC(7,5H,SIGN$VALU);
          /* REAL CONSTANT */
          CALL SET$TYP$N$LOC(7,6H,SIGN$VALU);
           /* STRING CONSTANT */
           CALL SET$TYP$N$LOC(7,7H,0);
        END; /* OF CASE */
      END;
      /* TYPE ENTRY */
      ;
      /* VARIABLE ENTRY */
      DO;
        IF SHR(FORM$FIELD(SP),7) THEN VARPARM = TRUE;
        PTRPTR = (SHR(FORM$FIELD(SP),3) AND FORMMASK);
        BASE$LOC(SP) = BASE; /* SYMBOL TABLE LOCATION OF
VARIABLE */
        CALL CASEPTRPTR(PTRPTR);
        CALL SET$TYP$N$LOC(7,PTRPTR,0);
      END;
      /* PROCEDURE ENTRY */
      ; /* NO SUCH THING EXISTS IN PASCAL */
      /* FUNCTION ENTRY */
      DO;
        IF FORM$FIELD(SP) = BUILT$IN$FUNC THEN /* BUILT IN
FUNCTION */
```

```
        DO;
          CALL SET$PAST$PN(8);
          IF BYTEPTR(0) <> 13H THEN
            IF BYTEPTR(0) <> 0FEH THEN
            DO:
              CALL CASEPTRPTR(BYTEPTR(0));
              TYPE$STACK(SP) = PTRPTR;
            END;
          APTRADDR = APTRADDR + 1;
          PARMNUM(SP) = BYTEPTR(0);
          PARMNUMLOC(SP) = APTRADDR + 1;
        END;
        ELSE DO;
          CALL SET$PAST$PN(16);
          CALL CASEPTRPTR(SHR(BYTEPTR(0),3) AND FORMMASK);
          CALL SET$TYP$N$LOC(10,PTRPTR,0);
          CALL SET$PAST$PN(7);
          PARMNUM(SP) = BYTEPTR(0);
          CALL SET$PAST$PN(8);
          PARMNUMLOC(SP) = ADDRPTR;
          CALL SET$PAST$PN(14);
          LABELSTACK(SP) = ADDRPTR;
        END:
        IF TOKEN <> 18 THEN READPARMS = TRUE;
      /* OTHERWISE, THIS WILL BE A FUNCTION ASSIGNMENT
STATEMENT */
        PARMNUMLOC(SP+2) = PARMNUMLOC(SP);
      END:
      /* FILE ENTRY */
      ;
      /* SCALAR ENTRY */
      DO;
        CALL SET$TYP$N$LOC(7,11H,0);
        APTRADDR = APTRADDR + 1;
        BASE$LOC(SP) = ADDRPTR;
      END;
    END: /* OF CASE */
 END SET$VAR$TYPE;



 /**************************************************************/
 /* LOAD$VARI - THIS PROCEDURE GENERATES THE */
 /* INTERMEDIATE CODE TO LOAD THE NEXT VARIABLE */
 /* ON THE EXECUTION STACK OF THE OBJECT FILE */
 /**************************************************************/ /* */
LOAD$VARI: PROC(PT) PUBLIC;
   DCL PT BYTE; /* PT REPRESENTS A STACK POINTER */
   EXP$STACK: PROC ;
        DCL A BYTE;
        DO CASE (TYPE$STACK(PT) AND 0FE);
          A = OPD$TYPE;
          A = ORD$TYPE;
          ;
```

```
          :
          A = BOOLEAN$TYPE;
          A = INTEGER$TYPE;
          A = UNSIGN$EXPON;
          A = STRING$TYPE;
          A = BOOLEAN$TYPE;
          A = INTEGER$TYPE;
          A = UNSIGN$EXPON;
          A = CHAR$TYPE;
        END; /* OF CASE */
        EXPRESS$STK(PT) = A;
     END EXP$STACK;
    LOAD: PROC(A, B, C);
        DCL (A, B, C) BYTE;
        /* CHECK IF LOADING A FUNCTION VALUE */
        IF (FORM$FIELD(PT) AND 7FH) <> FUNC$ENTRY THEN
        DO;
          CALL GENERATE(A);
          CALL GENERATE(B);
          IF SHR(TYPE$STACK(PT), 6) THEN /* ACCESSING ARRAY
*/
            CALL GENERATE(SUB);
          ELSE CALL GENERATE(C);
          IF A = LDIB THEN /* LOAD REST OF BCD NUMBER */
          DO PTRPTR = 2 TO (BCDNUM(Ø)/2);
               APTRADDR = APTRADDR - 2;
               CALL GENERATE(BYTEPTR(Ø));
               CALL GENERATE(HIGH(ADDRPTR));
          END;
          IF SHR(FORM$FIELD(PT),7) THEN /* VARIABLE
PARAMETER */
            CALL GENERATE(LODI);
        END;
        ELSE CALL GEN$ADDR(PRO,LABELSTACK(MP));
        CALL EXP$STACK;
     END LOAD;
    IF READSTMT THEN RETURN; /* GOING TO READ THIS VALUE */
    IF READPARMS THEN
    DO; /* READING A SUBROUTINE'S PARAMETERS */
      IF (TOKEN <> 12) AND (TOKEN <> 8) THEN READPARMS =
FALSE;
      /* THIS MEANS THIS PARAMETER IS AN EXPRESSION THAT
MUST BE
        EVALUATED. AFTER EVALUATION, READPARMS WILL BE SET
TO TRUE. */
      ELSE DO;
        CALL ASSIGN$PARMS;
        CALL EXP$STACK;
        RETURN;
      END;
    END;
    /* IF LOADING A FUNCTION VALUE, GO TO THE CASE STATEMENT
*/
    IF (FORM$FIELD(MP) AND 7FH) <> FUNC$ENTRY THEN
```

```
    DO;
      IF ((TYPE$STACK(PT) > 08H) AND (TYPE$STACK(PT) < 11H))
OR
      ((TYPE$STACK(PT) AND 40H) = 40H) THEN /* IN CASE OF
ARRAYS */
        CALL GENERATE(LITA); /* GOING TO LOAD A PRT ADDR */
      ELSE APTRADDR = PRTADDR(PT); /* GOING TO LOAD A
CONSTANT */
    END;
    DO CASE (TYPE$STACK(PT) AND 0FH); /*0*/ /* ORD VARIABLE
*/
          CALL LOAD(LOW(PRTADDR(PT)),HIGH(PRTADDR(PT)),LOD);
/*1*/ /* ORD CONSTANT */
          ; /*2*/ ; /*3*/ ; /*4*/ /* BOOLEAN CONSTANT */
          CALL LOAD(LDII,BYTEPTR(0),NOP); /*5*/ DO; /*
INTEGER CONSTANT */
            CALL LOAD(LDII,BYTEPTR(0),HIGH(ADDRPTR));
            IF TYPE$STACK(PT) = 35H THEN CALL GENERATE(NEGI);
          END; /*6*/ DO; /* BCD CONSTANT */
            CALL LOAD(LDIB,BYTEPTR(0),HIGH(ADDRPTR));
            IF TYPE$STACK(PT) = 86H THEN CALL GENERATE(NEGB);
          END; /*7*/ DO; /* STRING CONSTANT */
            CALL GENERATE(LDSI);
            TEMPBYTE = BYTEPTR(0); /* LENGTH OF STRING */
            DO PTRPTR = 0 TO TEMPBYTE;
              CALL GENERATE(APTRADDR + PTRPTR);
            END;
          END; /*8*/ /* BOOLEAN VARIABLE */
          CALL LOAD(LOW(PRTADDR(PT)),HIGH(PRTADDR(PT)),LOD);
/*9*/ /* INTEGER VARIABLE */
          CALL LOAD(LOW(PRTADDR(PT)),HIGH(PRTADDR(PT)),LODI);
/*A*/ /* REAL VARIABLE */
          CALL LOAD(LOW(PRTADDR(PT)),HIGH(PRTADDR(PT)),LODB);
/*B*/ /* CHARACTER VARIABLE */
          CALL LOAD(LOW(PRTADDR(PT)),HIGH(PRTADDR(PT)),LOD);
    END; /* OF CASE */
 END LOAD$VARI;


 /*************************************************************/
 /* ASSIGN$VARI - THIS PROCEDURE GENERATES THE */
 /* INTERMEDIATE CODE TO LOAD THE LEFT SIDE OF */
 /* AN ASSIGNMENT STATEMENT ON THE EXECUTION */
 /* STACK AND STORES A RESULT AT THAT LOCATION. */
 /*************************************************************/
ASSIGN$VARI: PROC(LS, STORE$TYPE) PUBLIC;
    DCL LS BYTE; /* LS IS THE LEFT SIDE OF ASSMT STMT */
    DCL (A, B, STORE$TYPE) BYTE; /* STORE$TYPE INDICATES
WHETHER
    TO DELETE OR LEAVE THE CURRENT VALUE AT THE TOP OF THE
STACK */
    IF (TYPE$STACK(LS) AND 40H) = 40H THEN
    DO;
```

```
      TYPE$STACK(LS) = (TYPE$STACK(LS) AND 0FFH);
      CALL GENERATE(XCHG);
    END;
    ELSE CALL GEN$ADDR(LITA,PRT$ADDR(LS));
    IF SHR(FORM$FIELD(LS),7) THEN /* CHECK FOR VAR PARAMETER
*/
      CALL GENERATE(LODI);
    DO CASE EXPRESS$STK(SP);
      /* CASE 0 - ORD TYPE */
      IF (TYPE$STACK(LS) <> 11H) AND (TYPE$STACK(LS) <> 10H)
THEN
        CALL ERROR('AT');
      ELSE A = 2;
      /* CASE 1 - INTEGER TYPE */
      IF TYPE$STACK(LS) = 09H THEN
         A = 1;
      ELSE DO;
        IF TYPE$STACK(LS) = 0AH THEN
        DO;
          CALL GENERATE(CNAI);
           A = 0;
        END;
        ELSE CALL ERROR('AT');
      END;
      /* CASE 2 - CHAR$TYPE */
      IF TYPE$STACK(LS) = 03H THEN
         A = 2;
      ELSE CALL ERROR('AT');
      /* CASE 3 - REAL TYPE */
      IF TYPE$STACK(LS) = 0AH THEN
         A = 0;
      ELSE CALL ERROR('AT');
      /* CASE 4 - STRING TYPE */
      A = 2;
      /* CASE 5 - BOOLEAN TYPE */
      IF TYPE$STACK(LS) = 06H THEN
         A = 2;
      ELSE CALL ERROR('AT');
    END; /* OF CASE */
    IF STORE$TYPE THEN A = A + 3;
    DO CASE A;
      B = STDB;
      B = STDI;
      B = STD;
      B = STOB;
      B = STOI;
      B = STO;
    END; /* OF CASE */
    CALL GENERATE(B);
 END ASSIGN$VARI;


 /*******************************************/
```

```
/* THIS PROCEDURE CHECKS THE TOP TWO */
/* VARIABLES ON THE EXECUTION STACK */
/* FOR PROPER TYPE. */
/****************************************/ CHK$EXPR$TYPE: PROC
BYTE PUBLIC;
    IF (EXPRESS$STK(SP) = EXPRESS$STK(MP)) AND
EXPRESS$STK(SP) <> 0H
      THEN RETURN TRUE;
    IF EXPRESS$STK(SP) = 1H THEN
    DO;
      IF EXPRESS$STK(MP) = 3H THEN
      DO;
        CALL GENERATE(CNVI); /* CONVERT INT TO BCD */
        EXPRESS$STK(SP) =3H;
        RETURN TRUE;
      END;
      ELSE RETURN FALSE;
    END;
    IF EXPRESS$STK(SP) =3H THEN
    DO;
      IF EXPRESS$STK(MP) = 1H THEN
      DO;
        CALL GENERATE(CN2I); /* CONVERT SECOND INT TO BCD */
        EXPRESS$STK(MP) = 3H;
        RETURN TRUE;
      END;
      ELSE RETURN FALSE;
    END;
    IF EXPRESS$STK(SP) = 0H THEN
    DO;
      IF EXPRESS$STK(MP) <> 0H THEN
        RETURN FALSE;
      ELSE DO;
        IF BASE$LOC(SP)=BASE$LOC(MP) THEN
          RETURN TRUE;
      END;
    END;
    RETURN FALSE;
 END CHK$EXPR$TYPE;


  /*****************************************************
   * COPY$STACKS - THIS PROCEDURE DUPLICATES THE *
   * STACK VALUES STORED AT ONE POINTER LOCATION*
   * AT ANOTHER SPECIFIED POINTER LOCATION. *
   *****************************************************/
COPY$STACKS: PROC(A, B) PUBLIC;
    DCL (A, B) BYTE;
    TYPE$STACK(A) = TYPE$STACK(B);
    PRT$ADDR(A) = PRT$ADDR(B);
    EXPRESS$STK(A) = EXPRESS$STK(B);
    FORM$FIELD(A) = FORM$FIELD(B);
    BASE$LOC(A) = BASE$LOC(B);
```

```
END COPY$STACKS;


  /*********************************************
   * GENERATE$BUILT$IN - THIS PROCEDURE *
   * GENERATES CODE FOR THE BUILT-IN *
   * FUNCTION. *
   *********************************************/ GEN$BUILT$IN:
PROC;
    APTRADDR = PARMNUMLOC(MP) - 2;
    IF (BYTEPTR(0) = 13H) OR (BYTEPTR(0) = 0F3H) THEN
      CALL COPY$STACKS(MP, SP-1);
    ELSE EXPRESS$STK(MP) = BYTEPTR(0);
    /* GENERATE THE NUMONIC CODE FOR THE BUILT IN FUNCTION
*/
    APTRADDR = APTRADDR - 1;
    DO CASE BYTEPTR(0);
      CALL GENERATE(ABS);
      CALL GENERATE(SQR);
      CALL GENERATE(SIN);
      CALL GENERATE(COS);
      CALL GENERATE(ARCTN);
      CALL GENERATE(EXP);
      CALL GENERATE(LN);
      CALL GENERATE(SQRT);
      CALL GENERATE(ODD);
      CALL GENERATE(EOLN);
      CALL GENERATE(EXF);
      CALL GENERATE(TRUNC);
      CALL GENERATE(ROUND);
      CALL GENERATE(ORD);
      CALL GENERATE(CHR);
      CALL GENERATE(SUCC);
      CALL GENERATE(PRED);
    END; /* OF CASE */
 END GEN$BUILT$IN;


  /*********************************************/
  /* WRITE$STRING - THIS PROCEDURE WRITES */
  /* A STRING TO THE INTERMED. CODE */
  /*********************************************/ WRITE$STRING:
PROC(NUMB) PUBLIC;
    DCL NUMB BYTE;
    CALL GENERATE(WRTS);
    CALL GENERATE(NUMB);
 END WRITE$STRING;


  /*********************************************/
  /* WRITE$VARIABLE - THIS PROCEDURE WILL */
```

```
  /* WRITE A VARIABLE TO THE CONSOLE VIA */
  /* THE INTERMED. CODE. */
  /*********************************************/ WRITE$VAR:
PROC(NUMB) PUBLIC;
    DCL NUMB BYTE; /* NUMBER OF WRITE PARAMS */
    IF NOT READPARMS THEN
      DO CASE EXPRESS$STK(MP);
        /* ORD TYPE */
        CALL GENERATE(WRT);
        /* INTEGER TYPE */
        CALL GENERATE(WRTI);
        /* CHAR TYPE */
        CALL GENERATE(WRTI);
        /* REAL TYPE */
        CALL GENERATE(WRTB);
        /* STRING TYPE */
        DO;
          CALL WRITE$STRING(NUMB);
          RETURN;
        END;
        /* BOOLEAN TYPE */
        CALL GENERATE(WRTI);
    END; /* CASE EXPRESS$STK(MP) */
    CALL GENERATE(NUMB);
 END WRITE$VAR;



  /**************************************************/
  /* READ$VARIABLE - THIS PROCEDURE GENERATES */
  /* THE INTERMEDIATE CODE TO READ A VARIABLE*/
  /* FROM THE CONSOLE. */
  /**************************************************/ READ$VAR:
PROC PUBLIC;
    IF (TYPE$STACK(SP) < 09H) OR (TYPE$STACK(SP) > 0BH) THEN
      CALL ERROR('IR');
    ELSE DO CASE (TYPE$STACK(SP) - 9); DO;/*CASE INTEGER*/
CALL GENERATE(RDVI); CALL GEN$ADDR(LITA,PRTADDR(SP)); CALL
GENERATE(STDI); END;/*CASE INTEGER*/ DO;/*CASE BCD*/ CALL
GENERATE(RDVB); CALL GEN$ADDR(LITA,PRTADDR(SP)); CALL
GENERATE(STDB); END;/*CASE BCD*/ DO;/*CASE BYTE*/ CALL
GENERATE(RDV); CALL GEN$ADDR(LITA,PRTADDR(SP)); CALL
GENERATE(STD); END;/*CASE BYTE*/
    END; /* CASE (TYPE$STACK(SP) - 9) */
 END READ$VAR;



  /**************************************************
   * B$I$PROCEDURE - THIS PROCEDURE IS CALLED *
   * UPON RECOGNITION OF A BUILT-IN PROCEDURE *
   * STATEMENT. *
   **************************************************/
B$I$PROCEDURE: PROC;
```

```
    BASE = BASELOC(MP);
    CALL SET$PAST$PN(7);
    IF BYTEPTR(Ø) < 22 THEN /* FILE HANDLING PROCEDURE */
      DO CASE (BYTEPTR(Ø) - 17);
        CALL GENERATE(PUT);
        CALL GENERATE(GET);
        CALL GENERATE(RESET);
        CALL GENERATE(REWRT);
        CALL GENERATE(PAGE);
        CALL GENERATE(NEW);
      END; /* OF CASE (BYTEPTR(Ø) - 17) */
    ELSE DO CASE (BYTEPTR(Ø) - 22); /* VARIABLE NUMBER OF
PARAMETERS */
        NEWSTMT = FALSE;
        DISPOSE$STMT = FALSE;
        READ$STMT = FALSE;
        READ$STMT = FALSE;
        WRITE$STMT = FALSE;
        DO;
          WRITE$STMT = FALSE;
          CALL GENERATE(DUMP);
        END;
      END; /* OF CASE (BYTEPTR(Ø) - 22) */
 END B$I$PROCEDURE;



  /***********************************************************
   * BREAK$LINKS - THIS PROCEDURE REMOVES THE *
   * SYMBOL TABLE LOCATIONS FROM THE HASH TABLE *
   * FOR THOSE IDENTIFIERS THAT WERE LOCAL TO *
   * THE CURRENT SCOPE; AND THE SCOPE POINTER IS *
   * DECRIMENTED BY ONE. *
   ***********************************************************/
BREAK$LINKS: PROC;
    DO WHILE SBTBLSCOPE > SCOPE(SCOPE$NUM - 1);
      BASE = SBTBLSCOPE;
      CALL SETADDRPTR(4);
      IF ((BYTEPTR(Ø) AND FORMMASK) = 7F) THEN
      DO;
        CALL SETADDRPTR(2);
        SBTBLSCOPE, BASE = ADDRPTR;
      END;
      ELSE DO;
        CALL SETADDRPTR(5);
        SYMHASH = BYTEPTR(Ø);
        CALL SETADDRPTR(Ø);
        HASHTABLE(SYMHASH) = ADDRPTR;
        CALL SETADDRPTR(2);
        SBTBLSCOPE, BASE = ADDRPTR;
      END;
    END;
    SBTBLSCOPE = SCOPE(SCOPE$NUM - 1);
 END BREAK$LINKS;
```

```
 /*************************************************************
  * SCOPE$BRANCH - THIS PROCEDURE GENERATES THE *
  * INTERMEDIATE CODE THAT PERMITS BRANCHING *
  * AROUND ANY CODE GENERATED FOR SUBROUTINES. *
  *************************************************************/
SCOPE$BRANCH: PROC PUBLIC;
    IF SCOPE$NUM > 1 THEN
    DO;
      APTRADDR = PARAMNUMLOC + 7;
      CALL GEN$ADDR(BRL,(ADDRPTR+1));
      CALL GEN$ADDR(LBL,ADDRPTR);
    END;
 END SCOPE$BRANCH;


 /*************************************************************
  * LABEL$MAKER - THIS PROCEDURE ENTERS ALL *
  * LABELS IN THE SYMBOL$TABLE. *
  *************************************************************/
LABEL$MAKER: PROC PUBLIC;
    IF TYPENUM = INTEGER$TYPE THEN
    DO;
      CALL ENTER$VAR$ID(0,SP,LABL$ENTRY);
      CALL ENTER$LABEL;
    END;
 END LABEL$MAKER;


 /*************************************************************
  * USER$TYPE - THIS PROCEDURE PERMITS THE *
  * PLACEMENT OF USER DEFINED TYPES IN THE *
  * SYMBOL TABLE. *
  *************************************************************/
USER$TYP: PROC(A) PUBLIC;
    DCL A BYTE;
    TYPE$LOCT=SBTBL;
    IF LOOKUP$ONLY(SP) THEN
      CALL ERROR('DT');
    CALL ENTER$VAR$ID(0,SP,TYPE$DCLE);
    IF NOT PRESENT THEN
    DO;
      CALL LIMITS(3);
      APTRADDR=SBTBL;
      BYTEPTR(0)=A;
      APTRADDR=APTRADDR+1;
      ADDRPTR=PARENT$TYPE;
      SBTBL=SBTBL+3;
    END;
 END USER$TYP;
```

```
/****************************************************** *
COUNT$PARA$BYTES - THIS PROCEDURE IS USED TO * * DETERMINE
THE NUMBER OF BYTES ASSOCIATED WITH * * THE PARAMETERS OF A
SUBROUTINE CALL. THIS * * INFORMATION IS NECESSARY TO ALLOW
PARAMETER * * MAPPING FROM THE EXECUTION STACK INTO THE PRT
* * BY THE SAVP OPERATION. *
*******************************************************/
COUNT$PARA$BYTES: PROC(NUM$OF$PARAS) ADDR;
  DCL TEMPVAL ADDR,
      (NUM$OF$PARAS,I) BYTE;
  TEMPVAL=0;
  DO I=1 TO NUM$OF$PARAS;
    CALL SET$PAST$PN(8);
    APTRADDR=ADDRPTR + ((I-1)*3);
    IF BYTEPTR(0)=0BH THEN ALLC$QTY=2;
    ELSE
      IF BYTEPTR(0)=1BH THEN
        ALLC$QTY=8;
      ELSE
        ALLC$QTY=1;
    TEMPVAL=TEMPVAL + ALLC$QTY;
  END;
  RETURN TEMPVAL;
 END COUNT$PARA$BYTES;


  /*****************************************************
   * GEN$FCN$HDR$SIZE - THIS PROCEDURE IS USED TO *
   * DETERMINE THE NUMBER OF BYTES ALLOCATED IN *
   * THE PRT FOR A FUNCTION NAME. *
   *****************************************************/
GEN$FCN$HDR$SIZE: PROC ADDR; CALL SET$PAST$PN(16); IF
BYTEPTR(0) = 0BH THEN
  RETURN 02H; ELSE
  IF BYTEPTR(0) = 1BH THEN
    RETURN 08H;
  ELSE
    RETURN 01H;
 END GEN$FCN$HDR$SIZE;


/****************************************************** *
PRO$FCN$BYTESIZE - THIS PROCEDURE RETURNS THE * * NUMBER OF
BYTES ALLOCATED IN THE PRT FOR A PRO- * * CEDURE OR FUNCTION
DECLARATION. THIS DATA IS * * REQUIRED TO ALLOW PARAMETER
MAPPING INTO THE PRT* * BY A SAVP OPERATOR. *
*******************************************************/
PRO$FCN$BYTESIZE: PROC ADDR; CALL SETADDRPTR(4); IF
BYTEPTR(0) = 04H THEN
```

```
  RETURN 02H; ELSE
  RETURN GEN$FCN$HDR$SIZE;
 END PRO$FCN$BYTESIZE;



   /*************************************************
    * SETSAVE$BLOCK - THIS PROCEDURE IS *
    * CALLED UPON DETERMINATION OF A SUBROUTINE *
    * BLOCK. IT INCRIMENTS THE PRT BY ONE LOCA- *
    * TION TO PERMIT THE INSERTION OF THE SBP== *
    * AND THIS ALLOWS FOR RECURSIVE CALLS. *
    *************************************************/
SETSAVE$BLOCK: PROC PUBLIC;
    DCL BYTE$COUNTER ADDR,
        COUNT BYTE;
    BYTE$COUNTER=0;
    LAST$SBTBL$ID = SBTBL;
    IF SCOPE$NUM > 1 THEN
    DO;
      BASE = PARMNUMLOC(SP - 5);
      CALL SET$PAST$PN(12);
      ADDRPTR = ALLOC$ADDR; /* SBP */
      ALLOC$ADDR = ALLOC$ADDR + 2;
      CALL SET$PAST$PN(7);
      TEMPADDR = BYTEPTR(0) AND 0FFH;
      CALL GEN$ADDR(LDII,TEMPADDR);
      CALL GEN$ADDR(LDII,PRO$FCN$BYTESIZE);
      BYTE$COUNTER=COUNT$PARA$BYTES(COUNT);
      CALL GEN$ADDR(LDII,BYTE$COUNTER);
      CALL SET$PAST$PN(10);
      CALL GEN$ADDR(LITA,ADDRPTR);
      CALL SET$PAST$PN(12);
      CALL GEN$ADDR(LITA,ADDRPTR);
      CALL GENERATE(SAVP);
    END;
 END SETSAVE$BLOCK;



   /*************************************************
    * HEAD$AND$BLOCK - UPON RECOGNITION OF A *
    * SUBROUTINE'S HEADING AND BLOCK, THIS *
    * PROCEDURE IS CALLED TO GENERATE REQUIRED *
    * CODE FOR UNSAVING THE SUBROUTINE'S *
    * PARAMETERS IN THE EVENT OF RECURSIVE CALLS.*
    *************************************************/
HEAD$N$BLK: PROC PUBLIC;
    BASE = PARMNUMLOC(MP);
    CALL SET$PAST$PN(12);
    CALL GEN$ADDR(LITA,ADDRPTR);
    CALL SET$PAST$PN(10);
    CALL GEN$ADDR(LITA,ADDRPTR);
    CALL GENERATE(UNSP);
```

```
    CALL BREAK$LINKS;
    BASE = PARMNUMLOC(MP);
    SCOPE$NUM=SCOPE$NUM-1;
    CALL GENERATE(RTN);
    CALL SET$PAST$PN(14);
    CALL GEN$ADDR(LBL,(ADDRPTR+1));
    TEMPADDR = 00H;
    CALL GEN$ADDR(LDII,TEMPADDR);
    CALL SET$PAST$PN(12);
    CALL GEN$ADDR(LITA,ADDRPTR);
    CALL GENERATE(STDI);
 END READ$N$BLK;


  /*************************************************************
   * FORWARD$SUBROUTINE - IN THE EVENT OF A *
   * FORWARD DEFINED SUBROUTINE, THE ALLOCATED *
   * SPACES IN THE PRT FOR THE ROUTINE AND ITS *
   * ASSOCIATED PARAMETERS ARE DE-ALLOCATED AND *
   * WILL BE REALLOCATED AT THE POINT OF THE *
   * SUBROUTINE´S DEFINITION. *
   *************************************************************/
FWD$SUBRTN: PROC PUBLIC;
    SCOPE$NUM = SCOPE$NUM - 1;
    APTRADDR = PARAMNUMLOC + 3;
    ALLOC$ADDR = ADDRPTR;
 END FWD$SUBRTN;


  /*************************************************************
   * GOT$PARAMETERS - THIS PROCEDURE IS CALLED *
   * ONCE ALL A SUBROUTINE´S PARAMETERS HAVE *
   * BEEN RECOGNIZED AND ENTERED IN THE SYMBOL *
   * TABLE. THE NUMBER OF PARAMETERS AND THEIR *
   * ASSOCIATED TYPE ARE THEN STORED IN THE *
   * SYMBOL TABLE. *
   *************************************************************/
GOT$PARAMS: PROC PUBLIC;
    APTRADDR = PARAMNUMLOC;
    BYTEPTR(0) = PARAMNUM;
    CALL ENTR$PRM$TYP;
 END GOT$PARAMS;


  /*************************************************************
   * SET$OP$TYPE - THIS PROCEDURE IS CALLED TO *
   * LOAD THE TYPE OF OPERATOR USED IN AN EX- *
   * PRESSION. *
   *************************************************************/
SET$OP$TYPE: PROC(A) PUBLIC;
    DCL A BYTE;
```

```
    TYPE$STACK(MP)=A;
 END SET$OP$TYPE;


  /*************************************************************
   * CALL$A$PROCEDURE - THIS PROCEDURE IS CALLED *
   * TO GENERATE INTERMEDIATE CODE UPON *
   * INVOKING A SUBROUTINE. THE NUMBER OF *
   * PARAMETERS REQUIRED IS ALSO CHECKED. *
   *************************************************************/
CALL$A$PROC: PROC(A) PUBLIC;
    DCL A BYTE; /* TRUE OR FALSE */
    READPARMS = FALSE;
    IF A THEN /* THE SUBROUTINE HAS PARAMETERS */
    DO;
      IF PARMNUM(MP) <> PARMNUM(SP-1) THEN
        CALL ERROR('PN');
    END;
    IF SHR(FORM$FIELD(MP),3) THEN
    DO;
      IF FORM$FIELD(MP) = 0DE THEN
        CALL GEN$BUILT$IN;
      ELSE CALL B$I$PROCEDURE;
    END;
    ELSE DO;
      IF FORM$FIELD(MP) = FUNC$ENTRY THEN
        CALL LOAD$VARI(MP);
      ELSE DO;
        CALL GEN$ADDR(PRO,LABELSTACK(MP));
        CALL GENERATE(DEL);
      END;
    END;
 END CALL$A$PROC;


  /*************************************************************
   * GOT$FUNCTION$TYPE - THIS PROCEDURE ENTERS *
   * THE TYPE OF THE FUNCTION INTO THE SYMBOL *
   * TABLE AND ALLOCATES A POSITION IN THE PRT *
   * FOR THE FUNCTION VALUE TO BE STORED IN. *
   *************************************************************/
GOT$FUNC$TYPE: PROC PUBLIC;
    BASE=PARMNUMLOC(MP);
    CALL SET$PAST$PN(16);
    BYTEPTR(0)=SHL(ALOCBASICTYP,3) OR VAR$ENTRY;
    CALL SET$PAST$PN(10);
    ALLOC$ADDR = ADDRPTR;
    ALLOC$ADDR = ALLOC$ADDR + ALLC$QTY;
 END GOT$FUNC$TYPE;
```

```
  /*************************************************************
   * END$PROGRAM - THIS PROCEDRE IS CALLED UPON *
   * RFCOGNITION OF THE END OF A PROGRAM. IT *
   * PRINTS OUT THE ERROR COUNT, CLOSES THE *
   * INTERMEDIATE FILE, WRITES THE SYMBOL TABLE *
   * FILE, AND INFORMS THE PROGRAMMER OF PROGRAM*
   * COMPILATION. *
   *************************************************************/
END$PROGRAM: PROC PUBLIC;
    CALL PRINT$ERROR;
    CALL PRINTCHAR(' ');
    CALL CRLF;
    IF NOT (ERRORCOUNT > 0) THEN
    DO;
      CALL GEN$ADDR(ALL,ALLOC$ADDR);
      CALL GENERATE(ENDP);
    END;
    CALL WRIT$INT$FILE;
    CALL MOVE$SBTBL;
    CALL CLOSE$INT$FIL;
    CALL PRINT(.(' COMPILATION COMPLETE.$ '));
    CALL MON3;
 END END$PROGRAM;


  /*************************************************************
   * ARRAY$DECLARE - THIS PROCEDURE DETERMINES *
   * AND STORES SYMBOLTABLE INFO ON ARRAYS. *
   * THIS PROCEDURE FAILS TO MAKE USE OF THE *
   * PARALLEL PARSE STACKS. *
   *************************************************************/
ARAY$DECLARE: PROC PUBLIC;
      DCL I BYTE;
      IF ARRY$PTR = -1 THEN ARRY$PTR=0;
      CALL ENTR$CPLX$TYP(17H);
      ARY$DM$ADR$PTR=ARY$DM$ADR$PTR-NUM$ARRY$DIM(ARRY$PTR);
      ARRY$BASE=BASE;
      CALL LIMITS((NUM$ARRY$DIM(ARRY$PTR)*4)+8);
      CALL SETADDRPTR(5); /*NUMBER OF DIMENSIONS*/
      BYTEPTR(0)=NUM$ARRY$DIM(ARRY$PTR);
      CALL SETADDRPTR(6); /*ADDRESS OF COMPONENT TYPE*/
      ADDR$PTR=TYPE$LOCT;
      CALL ALLC$OFFSET(TYPE$LOCT);
      BASE=ARRY$BASE;
      CALL SETADDRPTR(8); /*TOTAL STORAGE REQUIRED*/
      ADDRPTR=ARRY$CTY(ARRY$PTR)*ALLC$QTY;
      CALL SETADDRPTR(10); /*COMPONENT TYPE*/
      BYTEPTR(0)=ALOCBASICTYP;
      /*************************************************************
       THE FOLLOWING CODE CALCULATES THE OFFSET AND DIS-
       PLACEMENT VECTORS FOR EACH ARRAY DECLARATION AS
       FOLLOWS:
```

```
    WHERE N = # DIMENSIONS IN THIS ARRAY,
          D(I) = DISPLACEMENT VECTOR FOR ITH ARRAY
          U(I) = UPPER BOUND OF ITH ARRAY
          L(I) = LOWER BOUND OF ITH ARRAY
          V = OFFSET FOR THIS ARRAY

    FOR I = N DOWNTO 1
      IF I = N THEN D(I) = 1 ELSE
          D(I) = (U(I+1)-L(I+1))*D(I+1)
      V = V - (L(I)*D(I))

  ****************************************************************/
  ARRY$OFFSET = Ø; /*INIT FOR ZERØ-ORIGIN*/
  SUBR$FORM = NUM$ARRY$DIM(ARRY$PTR);
  DISP$VEC(SUBR$FORM) = 1;
  ARRY$OFFSET =
      ARRY$OFFSET -
      (ARRY$DIM$LOWVAL(ARY$DM$ADR$PTR + SUBR$FORM) *
       DISP$VEC(SUBR$FORM)):
  SUBR$FORM = SUBR$FORM - 1;
  DO WHILE SUBR$FORM > Ø;
    DISP$VEC(SUBR$FORM) =
       (ARRY$DIM$HIVAL(ARY$DM$ADR$PTR + SUBR$FORM -1) -
        ARRY$DIM$LOWVAL(ARY$DM$ADR$PTR + SUBR$FORM +1)
+1) *
        DISP$VEC(SUBR$FORM +1);
    ARRY$OFFSET =
        ARRY$OFFSET -
        (ARRY$DIM$LOWVAL(ARY$DM$ADR$PTR + SUBR$FORM) *
         DISP$VEC(SUBR$FORM));
    SUBR$FORM = SUBR$FORM - 1:
    END; /*DO WHILE*/
  CALL SETADDRPTR(11); /*OFFSET*/
  ADDRPTR = ARRY$OFFSET * ALLC$QTY;
  CALL SETADDRPTR(13); /*ADDRESS OF DIMENSION 1*/
  DO I=1 TO NUM$ARRY$DIM(ARRY$PTR);
    ADDRPTR = ARRY$DIMEN(ARY$DM$ADR$PTR + I);
    APTRADDR = APTRADDR + 2; /*DISP VECTOR FOR THIS
DIMEN*/
    ADDRPTR = DISP$VEC(I) * ALLC$QTY;
    APTRADDR = APTRADDR + 2; /*SET-UP FOR NEXT DIM*/
    END;
  TYPE$LOCT=BASE;
  SBTBL=SBTBL+((NUM$ARRY$DIM(ARRY$PTR)*4)+8);
 ARRY$PTR=ARRY$PTR-1;
END ARAY$DECLARE;


 /*************************************************************
  * FIND$RELOP - THIS PROCEDURE DETERMINES *
  * WHAT MNEUMONIC SHOULD BE GENERATED FOR ANY *
  * RELATIONAL OPERATOR. *
  *************************************************************/
```

```
FIND$RELOP: PROC PUBLIC;
    DCL A BYTE;
    DO CASE (TYPE$STACK(MPP1)-8);
      A = EQLI;
      A = NEQI;
      A = LEQI;
      A = GEQI;
      A = LSSI;
      A = GRTI;
      IF EXPRESS$STK(SP) <> ORD$TYPE THEN CALL ERROR('CE');
      ELSE A = XIN;
    END; /* CASE (TYPE$STACK(MPP1)-8) */
    DO CASE EXPRESS$STK(SP);
      /* ORD TYPE */
      IF (A = LSSI) OR (A = GRTI) THEN CALL ERROR('CE');
      ELSE IF A <> XIN THEN A = A + 19;
      /* INTEGER TYPE */
      ; /* NO OFFSET REQUIRED */
      /* CHAR TYPE */
      ; /* NO OFFSET REQUIRED */
      /* REAL TYPE */
      A = A + 7;
      /* STRING TYPE */
      A = A + 13;
      /* BOOLEAN TYPE */
      ; /* NO OFFSET REQUIRED */
    END; /* OF CASE EXPRESS$STK(SP) */
    CALL GENERATE(A);
    EXPRESS$STK(MP) = BOOLEAN$TYPE;
 END FIND$RELOP;



 END SYNTH1;
```

```
$PAGEWIDTH(80) TITLE('SYNTH2 - PRODUCTION CASE STATEMENTS')

SYNTH2: DO;

DECLARE LIT LITERALLY 'LITERALLY', EXT LIT 'EXTERNAL',
        DCL LIT 'DECLARE',
        POS LIT '0',
        NEG LIT '1',
        PROC LIT 'PROCEDURE',
        TRUE LIT '1',
        ADDR LIT 'ADDRESS',
        FALSE LIT '0',
        FOREVER LIT 'WHILE TRUE',
        STATESIZE LIT 'ADDRESS',
        BUILT$IN$PROC LIT '0CH',
        CONS$STR$TYPE LIT '3',
        CONS$NUM$TYPE LIT '0',
        CONS$IDENT$TYPE LIT '1',
        CONS$SIDENT$TYPE LIT '2'; DCL
        IDENTSIZE LIT '32', /* MAX IDENTIFIER SIZE - 1 */
        VARCSIZE LIT '100',/* SIZE OF VARC STACK*/
        PSTACKSIZE LIT '48', /* SIZE OF PARSE STACKS */
        HASHTBLSIZE LIT '128',/* SIZE OF HASHTABLE */
        BCDSIZE LIT '8', /* BYTES USED FOR BCD VALUES */
        MAX$NEST LIT '3', /* MAXLEVEL OF NESTS FOR TYPES */
        ARRY$NEST LIT '4', /* MAX NESTING LEVEL FOR ARRAYS
*/
        MAX$ARRY$DIM LIT '5', /* MAX ARRY DIMENSIONS */

/* FORM ENTRIES */
        LABL$ENTRY LIT '0',
        CONS$ENTRY LIT '1',
        TYPE$ENTRY LIT '2',
        VAR$ENTRY LIT '3',
        PROC$ENTRY LIT '4',
       FUNC$ENTRY LIT '5',
        FILE$ENTRY LIT '6',
        TYPE$DCLE LIT '7',

/* NUMBER TYPES */
        ORD$TYPE LIT '0',
        INTEGER$TYPE LIT '1',
        UNSIGN$EXPON LIT '3',
        STRING$TYPE LIT '4',
        BOOLEAN$TYPE LIT '5';

/* MANY OF THE FOLLOWING VARIABLES CAN BE REPLACED BY MAKING
           USE OF THE PARALLEL PARSE STACKS */
 DCL NUM$ARRY$ELMTS(25) ADDR, ARRY$DIM$LOWVAL(25) ADDR
PUBLIC, ARRY$DIM$HIVAL(25) ADDR PUBLIC, TEMP$BASE ADDR,
EXP$CTR BYTE, EXP$CTR1 BYTE, DISP$VEC(25) ADDR PUBLIC,
```

```
ARRY$OFFSET ADDR PUBLIC,
        SIGNTYPE BYTE EXT,
        VECPTR BYTE EXT,
        TYPENUM BYTE EXT,
        CONST$PTR BYTE EXT,
        STARTBDOS ADDR,/*ADDR OF PTR TO TOP OF BDOS*/
        TYPE$ADDR ADDR EXT,
        TYPE$LOCT ADDR EXT,
        VAR$PTR BYTE EXT,
        VAR$PARM$PTR BYTE EXT,
        ALOCBASICTYP BYTE EXT,
        ARRY$QTY(MAX$ARRY$DIM) ADDR EXT,
        VAR$BASE(10) ADDR EXT,
        ALLC$QTY ADDR EXT,
        TYPE$ORD$NUM BYTE EXT,
        PARENT$TYPE ADDR EXT,
        CONST$INDX BYTE EXT,
        LOOKUP$ADDR ADDR EXT,
        CONST$PN$PTR BYTE EXT,
        ARRY$PTR BYTE EXT,
        ARRY$DIM$PTR BYTE EXT,
        PTRPTR BYTE EXT,
        TAG$FD(MAX$NEST) BYTE EXT,
        VAR$CAS$TP(MAX$NEST) ADDR EXT,
        VAR$CAS$VAL(MAX$NEST) ADDR EXT,
        REC$NST BYTE EXT,
        RECORD$PTR BYTE EXT,
        REC$ADDR(10) ADDR EXT,
        REC$PAR$ADR(MAX$NEST) ADDR EXT,
        VARIANT$PART(MAX$NEST) BYTE EXT,
        FXD$OFST$BSE(MAX$NEST) ADDR EXT,
        VAR$OFST$BSE(MAX$NEST) ADDR EXT,
        CUR$OFST(MAX$NEST) ADDR EXT,
        NUM$ARRY$DIM(MAX$ARRY$DIM) BYTE EXT,
        ARRY$DIMEN(25) ADDR EXT,
        ARY$DM$ADR$PTR BYTE EXT,

/* CASE STATEMENT VARIABLES */
        CASE$STK(12) BYTE EXT,/* NUMBER OF STMTS IN CURRENT
CASE */
        CASE$COUNT BYTE EXT; /* LEVEL OF CASE STMTS */

/* GLOBAL VARIABLES */ DCL BCDNUM(BCDSIZE) BYTE EXT,
        SCOPE(10) ADDR EXT,
        SCOPE$NUM BYTE EXT,
        TEMPBYTE BYTE EXT,
        PRODUCTION BYTE EXT,
        PRV$SBT$ENTRY ADDR EXT;

/* COMPILER TOGGLES */ DCL
         LIST$PROD BYTE EXT,
         DEBUG$LN BYTE EXT,

/* COUNTERS */
```

```
        LABLCOUNT ADDR EXT,/* COUNTS NUMBER OF LABELS */

/* FLAGS USED DURING CODE GENERATION */
        CASE$STMT BYTE EXT,/* IN CASE STATEMENT */
        WRITE$STMT BYTE EXT,/* IN WRITE STATEMENT */
        READ$STMT BYTE EXT,/* IN READ STATEMENT */
        NEW$STMT BYTE EXT,/* GETS NEW RECORD */
        DISPOSE$STMT BYTE EXT,/* DISPOSES OF RECORD */
        VARPARM BYTE EXT, /* FORMAL PARAM IS VARIABLE
TYPE*/
        READPARMS BYTE EXT, /* READING ACTUAL PARAMETERS */
        PRESENT BYTE EXT, /* IDENTIFIER IS IN SYMBOL TABLE
*/
        SIGN$FLAG BYTE EXT, /* SET WHEN SIGN PRECEDES ID */

/* GLOBAL VARIABLES USED BY THE SCANNER */
        CONT BYTE EXT,/* INDICATES FULL ACCUM--STILL MORE
*/
        ACCUM(IDENTSIZE) BYTE EXT,/* HOLDS CURRENT TOKEN */

/* GLOBAL VARIABLES USED IN SYMBOL TABLE OPERATIONS */
BUILT$IN$TBL(10) BYTE EXT,
        BASE ADDR EXT,/* BASE LOCATION OF ENTRY */
        SBTBL ADDR EXT,/* CURRENT TOP OF SYMBOL TABLE */
        APTRADDR ADDR EXT,/* UTILITY VARIABLE TO ACCESS
SBTBL */
        ADDRPTR BASED APTRADDR ADDR,/* CURRENT 2 BYTES
POINTED AT */
        BYTEPTR BASED APTRADDR BYTE,/* CURRENT BYTE POINTED
AT */
        PRINTNAME ADDR EXT;/* SET PRIOR TO LOOKUP OR ENTER
*/

/* PARSER VARIABLES */
 DCL
       PARMNUM(PSTACKSIZE) BYTE EXT, /* MAINTAINS NUMBER OF
PARAMETERS ASSOCIATED WITH A SUBROUTINE */
       LABELSTACK(PSTACKSIZE) ADDR EXT,/* TRACKS STATEMENT
LABELS */
       PARMNUMLOC(PSTACKSIZE) ADDR EXT,/* MAINTAINS THE
LOCATION IN SYMBOL TBL WHERE PARAMETER INFO STORED */
       BASE$LOC(PSTACKSIZE) ADDR EXT, /* STORES THE SYMBOL
TABLE ADDRESS OF THE PERTINENT ENTRY */
       FORM$FIELD(PSTACKSIZE) BYTE EXT,/* STORES THE FORM
FIELD OF SCANNED IDENTIFIERS */
       TYPE$STACK(PSTACKSIZE)BYTE EXT,/* HOLDS A VARIABLE'S
TYPE */
       EXPRESS$STK(PSTACKSIZE)BYTE EXT./* CONTAINS THE
TYPES OF THE EXPRESSION COMPONENTS */
       PRT$ADDR(PSTACKSIZE) ADDR EXT, /* STORES AN
IDENTIFIER'S PRT LOCATION */
       PARAMNUM BYTE EXT,
       (SP,MP,MPP1) BYTE EXT;
```

```
/* MNEMONICS FOR PASCAL-SM MACHINE */
 DCL
 NOP    LIT  '0',ENDP  LIT  '1',LBL   LIT  '2',LDIB  LIT  '3',
 LDII   LIT  '4',PRO   LIT  '5',RTN   LIT  '6',SAVP  LIT  '7',
 UNSP   LIT  '8',CNVB  LIT  '9',CNVI  LIT '10',ALL   LIT '11',
 LITA   LIT '12',ADDB  LIT '13',ADDI  LIT '14',SUBB  LIT '15',
 SUBI   LIT '16',MULB  LIT '17',MULI  LIT '18',DIVB  LIT '19',
 DIVI   LIT '20',MODX  LIT '21',EQLI  LIT '22',NEQI  LIT '23',
 LEQI   LIT '24',GEQI  LIT '25',LSSI  LIT '26',GRTI  LIT '27',
 XIN    LIT '28',EQLB  LIT '29',NEQB  LIT '30',LEQB  LIT '31',
 GEQB   LIT '32',LSSB  LIT '33',GRTB  LIT '34',EQLS  LIT '35',
 NEQS   LIT '36',LEQS  LIT '37',GEQS  LIT '38',LSSS  LIT '39',
 GRTS   LIT '40',EQSET LIT '41',NEQST LIT '42',INCL1 LIT '43',
 INCL2  LIT '44',NEGB  LIT '45',
 NEGI   LIT '46',COMB  LIT '47',COMI  LIT '48',NOTX  LIT '49',
 ANDX   LIT '50',BOR   LIT '51',STOB  LIT '52'.STOI  LIT '53',
 STO    LIT '54',STDB  LIT '55',STDI  LIT '56',STD   LIT '57',
 UNION  LIT '58',STDIF LIT '59',ISEC  LIT '60',CNAI  LIT '61',
 BRL    LIT '62',BLC   LIT '63',CN2I  LIT '64',MKSET LIT '65',
 XCHG   LIT '66',PARM  LIT '67',PARMV LIT '68',PARMX LIT '69',
 INC    LIT '70',DEC   LIT '71',DEL   LIT '72',WRT   LIT '73',
 SUB    LIT '74',LDSI  LIT '75',KASE  LIT '76',LOD   LIT '77',
 LODB   LIT '78',LODI  LIT '79',RDVB  LIT '80',RDVI  LIT '81',
 RDVS   LIT '82',WRTB  LIT '83',WRTI  LIT '84',WRTS  LIT '85',
 DUMP   LIT '86',ABS   LIT '87',SQR   LIT '88',SIN   LIT '89',
 COS    LIT '90',ARCTN LIT '91',EXP   LIT '92',LN    LIT '93',
 SQRT   LIT '94',ODD   LIT '95',EOLN  LIT '96',EXF   LIT '97',
 TRUNC  LIT '98',ROUND LIT '99',ORD   LIT'100',CHR   LIT'101',
 SUCC   LIT'102',PRED  LIT'103',SEEK  LIT'104',PUT   LIT'105',
 GET    LIT'106',RESET LIT'107',REWRT LIT'108',PAGE  LIT'109',
 NEW    LIT'110',DISPZ LIT'111',FWD   LIT'112',XTRNL LIT'113',
 RDV    LIT'114';

SCANNER:PROC EXT;
 END SCANNER;

PRINT$PROD: PROC EXT;
 END PRINT$PROD;

ERROR: PROC(ERRCODE) EXT; DCL ERRCODE ADDR;
 END ERROR;

/* EXTERNAL PROCEDURES FROM SYMBOL.SRC */
 GENERATE: PROC(OBJCODE) EXT; DCL OBJCODE BYTE;
 END GENERATE;

GEN$ADDR: PROC(A,B) EXT; DCL A BYTE; DCL B ADDR;
 END GEN$ADDR;

SETADDRPTR: PROC(OFFSET) EXT;
    DCL OFFSET BYTE;
 END SETADDRPTR;

SET$PAST$PN: PROC(OFFSET) EXT;
```

```
    DCL OFFSET BYTE;
 END SET$PAST$PN;

CHK$PRT$NAME: PROC(A) BYTE EXT;
   /* A IS OFFSET FROM BASE TO PRINTNAME */
    DCL N BASED PRINTNAME BYTE;
    DCL (LEN,A) BYTE;
 END CHK$PRT$NAME;

LOOKUP$PN$ID: PROC(A,ID$ENTRY) BYTE EXT;
    DCL (A,ID$ENTRY) BYTE;
 END LOOKUP$PN$ID;

LIMITS: PROC(COUNT) EXT;
    DCL COUNT BYTE;
 END LIMITS;

ENTER$VAR$ID: PROC(A,B,ID$ENTRY) EXT;
    DCL (A,B,ID$ENTRY) BYTE;
 END ENTER$VAR$ID;

ALTER$PRT$LOC: PROC EXT;
    DCL (I,P) BYTE;
 END ALTER$PRT$LOC;

ENTER$SUBRTN: PROC(A,B,ID$ENTRY) EXT;
    DCL (A,B,ID$ENTRY) BYTE;
 END ENTER$SUBRTN;

LOOKUP$ONLY: PROC(A) BYTE EXT;
    DCL A BYTE;
 END LOOKUP$ONLY;

CONVRTBCD: PROC(A,B) EXT;/* A=SP/MP/MPP1, B=POS/NEG */
    DCL (I,J,DFLAG,EFLAG,SFLAG,A,B,N BASED PRINTNAME) BYTE;
 END CONVRTBCD;

CONVERTI: PROC(A,B) ADDRESS EXT;
    DCL (I,A,B,N BASED PRINTNAME) BYTE;
    DCL NUM ADDR;
 END CONVERTI;

CONVRT$CONST: PROC(A) EXT;/* A=POS,NEG */ DCL A BYTE;
 END CONVRT$CONST;

ENTR$CONS$NTRY: PROC EXT;
    DCL IXINDEX BYTE;
 END ENTR$CONS$NTRY;

ENTR$STR$TYP: PROC(A) EXT;
    DCL A BYTE;
 END ENTR$STR$TYP;

STORE$CONST: PROC EXT;
```

```
 END STORE$CONST;

/* EXTERNAL PROCEDURE DECLARATIONS FROM SYN1.SRC */

ORD$HI$LOW$CHK: PROC EXT;
 END ORD$HI$LOW$CHK;

ENTR$SUB$NTRY: PROC EXT;
 END ENTR$SUB$NTRY;

ALLC$OFFSET: PROC(A) EXT;/* TYPE$LOCT */
    DCL A ADDR;
    DCL (ALLC$FORM,B) BYTE;
 END ALLC$OFFSET;

AL$NDX$OFFSET: PROC ADDR EXT;
   DCL A ADDR,B BYTE;
 END AL$NDX$OFFSET;

ALLOC$VARS: PROC EXT;
 END ALLOC$VARS;

CASE$PTRPTR: PROC(A) EXT;
    DCL A BYTE;
 END CASE$PTRPTR;

SET$VAR$TYPE: PROC EXT;
 END SET$VAR$TYPE;

LOAD$VARI: PROC(PT) EXT;
    DCL PT BYTE; /* PT REPRESENTS A STACK POINTER */
 END LOAD$VARI;

ASSIGN$VARI: PROC(LS, STORE$TYPE) EXT;
    DCL LS BYTE; /* LS IS THE LEFT SIDE OF ASSMT STMT */
    DCL (A, B, STORE$TYPE) BYTE; /* STORE$TYPE INDICATES
WHETHER
    TO DELETE OR LEAVE THE CURRENT VALUE AT THE TOP OF THE
STACK */
 END ASSIGN$VARI;

CHK$EXPR$TYPE: PROC BYTE EXT;
 END CHK$EXPR$TYPE;

COPY$STACKS: PROC(A, B) EXT;
    DCL (A, B) BYTE;
 END COPY$STACKS;

WRITE$VAR: PROC(NUMB) EXT;
    DCL NUMB BYTE; /* NUMBER OF WRITE PARAMS */
 END WRITE$VAR;

READ$VAR: PROC EXT;
 END READ$VAR;
```

```
SCOPE$BRANCH: PROC EXT;
 END SCOPE$BRANCH;

LABEL$MAKER: PROC EXT;
 END LABEL$MAKER;

USER$TYP: PROC(A) EXT;
    DCL A BYTE;
 END USER$TYP;

SETSAVE$BLOCK: PROC EXT;
 END SETSAVE$BLOCK;

HEAD$N$BLK: PROC EXT;
 END HEAD$N$BLK;

FWD$SUBRTN: PROC EXT;
 END FWD$SUBRTN;

GOT$PARAMS: PROC EXT;
 END GOT$PARAMS;

SET$OP$TYPE: PROC(A) EXT;
    DCL A BYTE;
 END SET$OP$TYPE;

CALL$A$PROC: PROC(A) EXT;
    DCL A BYTE; /* TRUE OR FALSE */
 END CALL$A$PROC;

GOT$FUNC$TYPE: PROC EXT;
 END GOT$FUNC$TYPE;

END$PROGRAM: PROC EXT;
 END END$PROGRAM;

ARAY$DECLARE: PROC EXT;
 END ARAY$DECLARE;

FIND$RELOP: PROC EXT;
 END FIND$RELOP;

$EJECT SYNTHESIZE: PROC PUBLIC;

IF LISTPROD THEN
  CALL PRINT$PROD;

DO CASE PRODUCTION;

/***                P R O D U C T I O N S                      ***/
*****************************************************************
/*   CASE 0 NOT USED      */  ;
/*   1   <PROGRAM> ::= <PROGRAM HEADING> <BLOCK> . _^_       */
```

```
    CALL END$PROGRAM;
/*   2              ^ <PROCEDURE HEADING> <BLOCK> . _^_   */
    CALL END$PROGRAM;
/*   3              ^ <FUNCTION HEADING> <BLOCK> . _^_    */
    CALL END$PROGRAM;
/*   4   <PROGRAM HEADING> ::= PROGRAM <PROG IDENT> (    */
/*   4                          <XFILE IDENT> ) ;        */
    DO;
      SCOPE$NUM = 0;
      SCOPE(SCOPE$NUM) = SBTBL;
      SCOPE$NUM = 1;
    END;
/*   5   <XFILE IDENT> ::= <FILE IDENT>                  */
     ;
/*   6                   ^ <XFILE IDENT> , <FILE IDENT>  */
     ;
/*   7   <PROG IDENT> ::= <IDENTIFIER>                   */
     ;
/*   8 <FILE IDENT>::= <IDENTIFIER>                      */
    CALL ENTER$VAR$ID(16,SP,FILE$ENTRY);
/*   9   <BLOCK> ::= <LDP><CDP><TDP><VDP><P&FDP><STMTP>  */
     ;
/*  10   <LDP> ::=                                       */
    CALL SCOPE$BRANCH;
/*  11            ^ LABEL <LABEL STRING> ;               */
    CALL SCOPE$BRANCH;
/*  12   <LABEL STRING> ::= <LABEL>                      */
    CALL LABEL$MAKER;
/*  13                    ^ <LABEL STRING> , <LABEL>     */
    CALL LABEL$MAKER;
/*  14   <LABEL> ::= <NUMBER>                            */
    IF TYPENUM <> INTEGER$TYPE THEN
      CALL ERROR('LS');
/*  15   <CDP> ::=                                       */
     ;
/*  16            ^ CONST <CONST DEF> ;                  */
     ;
/*  17   <CONST DEF> ::= <IDENT CONST DEF>               */
     ;
/*  18                 ^ <CONST DEF> ; <IDENT CONST DEF> */
     ;
/*  19   <IDENT CONST DEF> ::= <IDENT CONST> = <CONSTANT> */
     CALL ENTR$CONS$NTRY;
/*  20   <IDENT CONST> ::= <IDENTIFIER>                  */
     DO;
       IF LOOKUP$ONLY(SP) THEN
         CALL ERROR('DC');
       CALL ENTER$VAR$ID(0,SP,CONS$ENTRY);
     END;
/*  21   <CONSTANT> ::= <NUMBER>                         */
     DO;
       CALL CONVRT$CONST(PCS);
       EXPRESS$STK(MP)=CONS$NUM$TYPE;
       VECPTR=VECPTR+1;
```

```
    END;
/*  22              ^ <SIGN> <NUMBER>                         */
    DO;
      IF SIGNTYPE=NEG THEN
        CALL CONVRT$CONST(NEG);
      ELSE CALL CONVRT$CONST(POS);
      EXPRESS$STK(MP)=CONS$NUM$TYPE;
      VECPTR=VECPTR+1;
      SIGN$FLAG = FALSE;
    END;
/*  23              ^ <CONSTANT IDENT>                        */
    DO;
      EXPRESS$STK(MP)=CONS$IDENT$TYPE;
      VECPTR=VECPTR+1;
      CALL STORE$CONST;
    END;
/*  24              ^ <SIGN> <CONSTANT IDENT>                 */
    DO;
      IF SIGNTYPE=NEG THEN
        EXPRESS$STK(MP)=CONS$SIDENT$TYPE;
      ELSE EXPRESS$STK(MP)=CONS$IDENT$TYPE;
      VECPTR=VECPTR+1;
      CALL STORE$CONST;
      SIGN$FLAG = FALSE;
    END;
/*  25              ^ <STRING>                                */
    DO;
      EXPRESS$STK(MP)=CONS$STR$TYPE;
      VECPTR=VECPTR+1;
      CALL STORE$CONST;
    END;
/*  26   <CONSTANT IDENT> ::= <IDENTIFIER>                    */
    ;
/*  27   <SIGN> ::= +                                         */
    DO;
     SIGN$TYPE = POS;
     SIGN$FLAG = TRUE;
    END;
/*  28             ^ -                                        */
    DO;
     SIGN$TYPE = NEG;
     SIGN$FLAG = TRUE;
    END;
/*  29   <TDP> ::=                                            */
    CASE$STMT=FALSE;
/*  30            ^ TYPE <TYPE DEF STRING> ;                  */
    CASE$STMT=FALSE;
/*  31   <TYPE DEF STRING> ::= <TYPE ID>                      */
    ;
/*  32                        ^ <TYPE DEF STRING> ; <TYPE ID> */
    ;
/*  33   <TYPE ID> ::= <TYPE IDS> = <TYPE>                    */
    DO;
      APTRADDR=TYPE$ADDR;
```

```
      ADDRPTR=TYPE$LOCT;
    END;
/*  34   <TYPE IDS> ::= <IDENTIFIER>                              */
    DO;
      IF LOOKUP$ONLY(SP) THEN
        CALL ERROR('DT');
      PARENT$TYPE=SBTBL;
      CALL ENTER$VAR$ID(78H,SP,TYPE$ENTRY);
      IF NOT PRESENT THEN
      DO;
        CALL LIMITS(2);
        TYPE$ADDR=SBTBL;
        SBTBL=SBTBL+2;
      END;
    END;
/*  35   <TYPE> ::= <SIMPLE TYPE>                                 */
    ;
/*  36              ^ <STRUCTURED TYPE>                           */
    ;
/*  37              ^ <POINTER TYPE>                              */
    ;
/*  38   <SIMPLE TYPE> ::= <TYPE IDENT>                           */
    ;
/*  39                   ^ ( <TIDENT STRING> )                    */
    ;
/*  40                   ^ <CONSTANT> .. <CONSTANT>               */
    CALL ENTR$SUB$NTRY;
/*  41   <TYPE IDENT> ::= <IDENTIFIER>                            */
    IF LOOKUP$PN$ID(SP,TYPE$ENTRY) THEN
      TYPE$LOCT=LOOKUP$ADDR;
    ELSE DO;
      CALL ERROR('TI');
      TYPE$LOCT=.BUILT$IN$TBL; /* INTEGER DEFAULT */
    END;
/*  42   <TIDENT STRING> ::= <IDENTIFIER>                         */
    DO;
      TYPE$ORD$NUM=0;
      CALL USER$TYP(TYPE$ORD$NUM);
    END;
/*  43                  ^ <TIDENT STRING> , <IDENTIFIER>          */
    DO;
      TYPE$ORD$NUM=TYPE$ORD$NUM+1;
      CALL USER$TYP(TYPE$ORD$NUM);
    END;
/*  44   <STRUCTURED TYPE> ::= <UNPACKED STRUCTURED TYPE>         */
    ;
/*  45                       ^ PACKED                             */
/*  45                         <UNPACKED STRUCTURED TYPE>         */
    ;
/*  46   <UNPACKED STRUCTURED TYPE> ::= <ARRAY TYPE>              */
    ;
/*  47                                ^ <RECORD TYPE>             */
    ;
/*  48                                ^ <SET TYPE>                */
```

```
      ;
/*   49                                ^ <FILE TYPE>          */
      ;
/*   50   <ARRAY TYPE> ::= ARRAY [ <INDEX TYPE STRING> ] OF*/
/*   50                   <COMPONENT TYPE>                 */
     CALL ARAY$DECLARE;
/*   51   <INDEX TYPE STRING> ::= <INDEX TYPE>            */
      DO;
        IF ARRY$PTR=ARRY$NEST-1 THEN
        DO;
          CALL ERROR('AN');
          ARY$DM$ADR$PTR = ARY$DM$ADR$PTR -
                           NUM$ARRY$DIM(ARRYY$PTR);
        END;
        ELSE ARRY$PTR= ARRY$PTR+1;
        ARRY$DIM$PTR=0;
        ARY$DM$ADR$PTR=ARY$DM$ADR$PTR+1;
        ARRY$QTY(ARRY$PTR) = AL$NDX$OFFSET;
        ARRY$DIMEN(ARY$DM$ADR$PTR) = TYPE$LOCT;
        TEMPBASE=BASE;
        BASE = TYPELOCT;
        CALL SETADDRPTR(7);
        ARRY$DIM$LOWVAL(ARY$DM$ADR$PTR) = ADDRPTR;
        CALL SETADDRPTR(9);
        ARRY$DIM$HIVAL(ARY$DM$ADR$PTR) = ADDRPTR;
        CALL SETADDRPTR(11);
        NUM$ARRY$ELMTS(ARY$DM$ADR$PTR) = ADDRPTR;
        BASE=TEMPBASE;
        NUM$ARRY$DIM(ARRY$PTR)=1;
      END;
/*   52                          ^ <INDEX TYPE STRING> ,     */
/*   52                            <INDEX TYPE>              */
      DO;
        IF ARRY$DIM$PTR=MAX$ARRY$DIM-1 THEN
          CALL ERROR('AD');
        ELSE ARRY$DIM$PTR=ARRY$DIM$PTR+1;
        ARY$DM$ADR$PTR=ARY$DM$ADR$PTR+1;
        ARRY$QTY(ARRY$PTR) = ARRY$QTY(ARRY$PTR) *
                             AL$NDX$OFFSET;
        ARRY$DIMEN(ARY$DM$ADR$PTR)=TYPE$LOCT;
        TEMPBASE=BASE;
        BASE=TYPE$LOCT;
        CALL SETADDRPTR(7);
        ARRY$DIM$LOWVAL(ARY$DM$ADR$PTR)=ADDRPTR;
        CALL SETADDRPTR(9);
        ARRY$DIM$HIVAL(ARY$DM$ADR$PTR)=ADDRPTR;
        CALL SETADDRPTR(11);
        NUM$ARRY$ELMTS(ARY$DM$ADR$PTR)=ADDRPTR;
        BASE=TEMPBASE;
        NUM$ARRY$DIM(ARRY$PTR)=NUM$ARRY$DIM(ARRY$PTR)-1;
      END;
/*   53   <INDEX TYPE> ::= <SIMPLE TYPE>                    */
      ;
/*   54   <COMPONENT TYPE> ::= <TYPE>                       */
```

```
    ;
/*  55   <RECORD TYPE> ::= <RECORD> <FIELD LIST>  END      */
    DO;
      VARIANT$PART(REC$NST)=FALSE;
      BASE,TYPE$LOCT=REC$PAR$ADR(REC$NST);
      IF VAR$CAS$VAL(REC$NST) <> 0 THEN
        CALL ERROR('IV');
      CALL SETADDRPTR(5);
      ADDRPTR=FXD$OFST$BSE(REC$NST);
      CALL SETADDRPTR(7);
      ADDRPTR= PRV$SBT$ENTRY;
      REC$NST=REC$NST-1;
    END;
/*  56   <RECORD> ::= RECORD                                */
    DO;
      REC$NST=REC$NST+1;
      APTRADDR,REC$PAR$ADR(REC$NST)=SBTBL;
      ADDRPTR=0000H; /*COLLISION ENTRY*/
      APTRADDR=APTRADDR+2;
      ADDRPTR=PRV$SBT$ENTRY;
      PRV$SBT$ENTRY=SBTBL;
      APTRADDR=APTRADDR+2;
      BYTEPTR=1FH;  /* FORM FOR RECORD */
      SBTBL=SBTBL+9;  /* ALLOW FOR REST OF ENTRY */
      /* INITIALIZE RECORD */
      VARIANT$PART(REC$NST),TAG$FD(REC$NST)=FALSE;
      FXD$OFST$BSE(REC$NST)=0000H;
      VAR$OFST$BSE(REC$NST)=0000H;
      CUR$OFST(REC$NST)=0000H;
      VAR$CAS$VAL(REC$NST)=0000H;
      RECORD$PTR=-1;
    END;
    ;
/*  57   <FIELD LIST> ::= <FIXED PART>                      */
/*  58                  ^ <FIXED PART> ; <VARIANT PART>     */
    ;
/*  59                  ^ <VARIANT PART>                    */
    ;
/*  60   <FIXED PART> ::= <RECORD SECTION>                  */
    ;
/*  61                  ^ <FIXED PART> ; <RECORD SECTION> */
    ;
/*  62  <RECORD SECTION> ::= <FIELD IDENT STRING> : <TYPE>*/
    DO;
       CALL ALLC$OFFSET(TYPE$LOCT);
      /* ALOCBASICTYP AND ALLC$QTY ARE SET */
       DO PTRPTR = 0 TO RECORD$PTR;
         BASE = REC$ADDR(PTRPTR);
         CALL SET$PAST$PN(9);
         BYTEPTR = ALOCBASICTYP;
         APTRADDR=APTRADDR+1;
         ADDRPTR=TYPE$LOCT;
         APTRADDR=APTRADDR+2;
         ADDRPTR=CUR$OFST(REC$NST);
```

```
          CUR$OFST(REC$NST)=CUR$OFST(REC$NST)
             + ALLOC$CTY;
        END;
        RECORD$PTR=0;
        IF FXD$OFST$BSE(REC$NST) < CUR$OFST(REC$NST)
        THEN FXD$OFST$BSE(REC$NST)=CUR$OFST(REC$NST);
      END;
/*   63                         ^                                    */
     ;
/*   64   <FIELD IDENT STRING> ::= <FIELD IDENT>                      */
     ;
/*   65                           ^ <FIELD IDENT STRING> ,            */
/*   65                             <FIELD IDENT>                     */
     ;
/*   66   <FIELD IDENT> ::= <IDENTIFIER>                              */
     DO;
       IF RECORD$PTR <> 10 THEN RECORD$PTR=RECORD$PTR+1;
       ELSE CALL ERROR('RN');
       REC$ADDR(RECORD$PTR)=SBTBL;
       CALL ENTER$VAR$ID(58H,SP,TYPE$DCLE);
       IF NOT PRESENT THEN DO;
         CALL LIMITS(7);
         APTRADDR=SBTBL;
         ADDRPTR=REC$PAR$ADR(REC$NST);
         SBTBL=SBTBL+7;
       END;
       IF VARIANT$PART(REC$NST) THEN
       DO;
         BASE=REC$ADDR(RECORD$PTR);
         CALL LIMITS(2);
         CALL SETADDRPTR(4);
         BYTEPTR=0DFF;
       END;
     END;
/*   67  <VARIANT PART> ::= CASE<TAG FIELD><TYPE IDENT> OF */
/*   67                       <VARIANT STRING>                        */
     ;
/*   68                     ^ CASE <TYPE IDENT> OF                    */
/*   68                       <VARIANT STRING>                        */
     ;
/*   69   <VARIANT STRING> ::= <VARIANT>                              */
     ;
/*   70                       ^ <VARIANT STRING> ; <VARIANT>*/
     ;
/*   71   <TAG FIELD> ::= <FIELD IDENT> :                            */
     TAG$FD(REC$NST)=TRUE;
/*   72  <VARIANT> ::=<CASE LABEL LIST> : (<FIELD LIST> )            */
     ;
/*   73                    ^                                          */
     ;
/*   74   <CASE LABEL LIST> ::= <CASE LABEL>                          */
     DO;
       LABELSTACK(SP) = LABLCOUNT;
       LABLCOUNT = LABLCOUNT + 2;
```

```
      CALL GEN$ADDR(KASE,ALLC$QTY);
      CALL GENERATE(LOW(LABELSTACK(SP)));
      CALL GENERATE(HIGH(LABELSTACK(SP)));
    END;
/*  75                    ^ <CASE LABEL LIST> , <CASE LABEL>*/
    DO;
      CALL GEN$ADDR(KASE,ALLC$QTY);
      CALL GENERATE(LOW(LABELSTACK(MP)));
      CALL GENERATE(HIGH(LABELSTACK(MP)));
    END;
/*  76   <CASE LABEL> ::= <CONSTANT>                              */
    IF CASE$STMT THEN
    DO;
      CASE$STK(CASE$COUNT) = CASE$STK(CASE$COUNT) + 1;
      DO CASE EXPRESS$STK(SP)  ;
        /* NUMBER */
         ALLC$QTY = CONVERTI(SP,POS);
        /* IDENTIFIER */
         DO;
           IF NOT LOOK$UP$ONLY(SP) THEN CALL ERROR('IT');
           ELSE DO;
             BASE = LOOKUP$ADDR;
             CALL SET$PAST$PN(7);
             ALLC$QTY = ADDRPTR;
           END;
         END;
        /* SIGNED IDENTIFIER */
        DO;
        END;
        /* STRING TYPE */
        ;
      END;          /*OF CASE*/
    END;
    ELSE
    DO;
     IF NOT VARIANT$PART(REC$NST) THEN
      DO;
       VARIANT$PART(REC$NST)=TRUE;
       VAR$CAS$TP(REC$NST)=TYPE$LOCT;
       VAR$CAS$VAL(REC$NST)=AL$NDX$OFFSET;
       CALL ALLC$OFFSET(TYPE$LOCT);
       IF TAG$FD(REC$NST) THEN
        DO;
          TAG$FD(REC$NST)=FALSE;
          BASE=REC$ADDR(RECORD$PTR);
          CALL SETADDRPTR(4);
          BYTEPTR=9FH;
          CALL SETADDRPTR(5);
          CALL SETADDRPTR(6+BYTEPTR);
          ADDRPTR=VAR$CAS$VAL(REC$NST);
          APTRADDR=APTRADDR+2;
          ADDRPTR=VAR$CAS$TP(REC$NST);
          APTRADDR=APTRADDR+2;
          ADDRPTR=CUR$OFST(REC$NST);
```

```
      CUR$OFST(REC$NST)=CUR$OFST(REC$NST)+ALLO$CTY;
       END;
      VAR$OFST$BSE(REC$NST)=CUR$OFST(REC$NST);
      FXD$OFST$BSE(REC$NST)=CUR$OFST(REC$NST);
     END;
   /* CALL COMPARE$CONST$VARIANT; */
   /*  CHECKS THE CASE LABLE WITH THE VARIANT TYPE */
    CUR$OFST(REC$NST)=VAR$OFST$BSE(REC$NST);
    VECPTR=VECPTR-1:
    CONST$PTR,CONST$INDX,CONST$PN$PTR=0;
   END;
/*  76   <SET TYPE> ::= SET OF <BASE TYPE>                        */
   CALL ENTR$STR$TYP(27H);
/*  78   <BASE TYPE> ::= <SIMPLE TYPE>                            */
   ;
/*  79   <FILE TYPE> ::= FILE OF <TYPE>                           */
   CALL ENTR$STR$TYP(2FH);
/*  80   <POINTER TYPE> ::= ^ <TYPE IDENT>                        */
   CALL ENTR$STR$TYP(37H);
/*  81   <VDP> ::=                                                */
   SCOPE(SCOPE$NUM) = SBTBL;
/*  82              ^ VAR <VAR DECLAR STRING> ;                   */
   SCOPE(SCOPE$NUM) = SBTBL;
/*  83   <VAR DECLAR STRING> ::= <VAR DECLAR>                     */
   ;
/*  84                               ^ <VAR DECLAR STRING> ;      */
/*  84                                 <VAR DECLAR>               */
   ;
/*  85   <VAR DECLAR> ::= <IDENT VAR STRING> : <TYPE>             */
   DO;
     CALL ALLOC$VARS;
   END;
/*  86   <IDENT VAR STRING> ::= <IDENTIFIER>                      */
   DO;
     VAR$PTR = 0;
     PARENT$TYPE,VAR$BASE(VARPTR) = SBTBL;
     CALL ENTER$VAR$ID(0,SP,VAR$ENTRY);
   END;
/*  87                              ^ <IDENT VAR STRING> ,        */
/*  87                                <IDENTIFIER>                */
   IF VARPTR <> 10 THEN
   DO;
     VAR$PTR = VAR$PTR + 1;
     VAR$BASE(VAR$PTR) = SBTBL;
     CALL ENTER$VAR$ID(0,SP,VAR$ENTRY);
   END;
   ELSE CALL ERROR('VN');
/*  88   <P&FDP> ::=                                              */
   CALL SETSAVE$BLOCK;
/*  89               ^ <PORF DECLAR>                              */
   CALL SETSAVE$BLOCK;
/*  90   <PORF DECLAR> ::= <PROC OR FUNCT> ;                      */
  ;
/*  91                         ^ <PORF DECLAR> <PROC OR FUNCT> ;*/
```

```
  ;
/*  92   <PROC OR FUNCT> ::= <PROCEDURE HEADING> <BLOCK>   */
   CALL HEAD$N$BLK;
/*  93                    ^ <PROCEDURE HEADING> <DIRECTIVE> */
    ;
/*  94                        ^ <FUNCTION HEADING> <BLOCK>    */
   CALL HEAD$N$BLK;
/*  95                    ^ <FUNCTION HEADING> <DIRECTIVE>   */
    ;
/*  96   <DIRECTIVE> ::= <IDENTIFIER>                        */
   IF NOT LOOKUP$ONLY(SP) THEN CALL ERROR('IT');
   ELSE DO;
     BASE = LOOKUP$ADDR;
     CALL SETADDRPTR(5);
     IF BYTEPTR = 21 THEN CALL FWD$SUBRTN;
   END;
/*  97   <PROCEDURE HEADING> ::= <PROC ID> ;                 */
   CALL GOT$PARAMS;
/*  98                           ^ <PROC ID> (                */
/*  98                          <FORMAL PARA SECT LIST> ) ; */
   CALL GOT$PARAMS;
/*  99   <PROC ID> ::= PROCEDURE <IDENTIFIER>                */
    DO;
      PARAMNUM = 0;
      CALL ENTER$SUBRTN(0,SP,PROC$ENTRY);
    END;
/* 100   <FORMAL PARA SECT LIST> ::= <FORMAL PARA SECT>      */
   CALL ALLOC$VARS;
/* 101                           ^ <FORMAL PARA SECT LIST> ;*/
/* 101                               <FORMAL PARA SECT>     */
   CALL ALLOC$VARS;
/* 102   <FORMAL PARA SECT> ::= <PARA GROUP>                 */
   ;
/* 103                          ^ VAR <PARA GROUP>           */
   DO;
     TEMPBYTE = VAR$PTR;
     DO VAR$PARM$PTR = 0 TO TEMPBYTE;
       BASE = VARBASE (VAR$PARM$PTR);
       CALL SETADDRPTR(4);
       BYTEPTR = BYTEPTR OR 80H;
     END;
   END;
/* 104                          ^ FUNCTION <PARA GROUP>      */
   DO;
     TEMPBYTE = VAR$PTR;
     DO VAR$PARM$PTR = 0 TO TEMPBYTE;
       BASE = VARBASE (VAR$PARM$PTR);
       CALL SETADDRPTR(4);
       BYTEPTR = FUNC$ENTRY OR 80H;
     END;
   END;
/* 105                        ^ PROCEDURE <PROC IDENT LIST>  */
   ;
/* 106   <PROC IDENT LIST> ::= <IDENTIFIER>                  */
```

```
    DO;
      VAR$PTR=0;
      PARAMNUM = PARAMNUM + 1;
      VAR$BASE(0)=SBTBL;
      CALL ENTER$SUBRTN(0,SP,PROC$ENTRY);
    END;
/* 107                      ^ <PROC IDENT LIST> , <IDENTIFIER>*/
    IF VAR$PTR <> 10 THEN
    DO;
      VAR$PTR=VAR$PTR+1;
      PARAMNUM = PARAMNUM + 1;
      VAR$BASE(VAR$PTR)=SBTBL;
      CALL ENTER$SUBRTN(0,SP,PROC$ENTRY);
    END;
    ELSE CALL ERROR('VN');
/* 108   <PARA GROUP> ::= <PARA IDENT LIST> : <TYPE IDENT>*/
    ;
/* 109   <PARA IDENT LIST> ::= <IDENTIFIER>                */
    DO;
      VAR$PTR=0;
      PARAMNUM = PARAMNUM + 1;
      VAR$BASE(0)=SBTBL;
      CALL ENTER$VAR$ID(0,SP,VAR$ENTRY);
    END;
/* 110                      ^ <PARA IDENT LIST> , <IDENTIFIER>*/
    IF VAR$PTR <> 10 THEN
    DO;
      VAR$PTR=VAR$PTR+1;
      PARAMNUM = PARAMNUM + 1;
      VAR$BASE(VAR$PTR)=SBTBL;
      CALL ENTER$VAR$ID(0,SP,VAR$ENTRY);
    END;
    ELSE CALL ERROR('VN');
/* 111  <FUNCTION HEADING> ::=<FUNCT ID> :<RESULT TYPE> ; */
    DO;
    CALL GOT$PARAMS;
    CALL GOT$FUNC$TYPE;
    END;
/* 112                            ^ <FUNCT ID> (           */
/* 112                          <FORMAL PARA SECT LIST> ) :  */
/* 112                              <RESULT TYPE> ;         */
    DO;
      CALL GOT$PARAMS;
      CALL GOT$FUNC$TYPE;
      CALL ALTER$PRT$LOC;
    END;
/* 113   <FUNCT ID> ::= FUNCTION <IDENTIFIER>               */
     DO;
       PARAMNUM = 0;
       CALL ENTER$SUBRTN(0,SP,FUNC$ENTRY);
     END;
/* 114   <RESULT TYPE> ::= <TYPE IDENT>                     */
    CALL ALLC$OFFSET(TYPELOCT);
/* 115   <STMTP> ::= <COMPOUND STMT>                        */
```

```
    ;
/* 116   <STMT> ::= <BAL STMT>                                     */
    ;
/* 117                 ^ <UNBAL STMT>                              */
    ;
/* 118                 ^ <LABEL DEF> <STMT>                        */
    ;
/* 119 <BAL STMT> ::=<IF CLAUSE><TRUE PART> ELSE<BAL STMT>*/
      CALL GEN$ADDR(LBL,(LABELSTACK(MP)+1));
/* 120                      ^ <SIMPLE STMT>                        */
    ;
/* 121   <UNBAL STMT> ::= <IF CLAUSE> <STMT>                       */
    CALL GEN$ADDR(LBL,LABELSTACK(MP));
/* 122                         ^ <IF CLAUSE> <TRUE PART> ELSE      */
/* 122                         <UNBAL STMT>                        */
    CALL GEN$ADDR(LBL,(LABELSTACK(MP)+1));
/* 123   <IF CLAUSE> ::= IF <EXPRESSION> THEN                      */
    DO;
      LABELSTACK(MP)=LABLCOUNT;
      LABLCOUNT=LABLCOUNT+2;
      IF EXPRESS$STK(MPP1) = BOOLEAN$TYPE THEN
      DO;
        CALL GENERATE(NOTX);
        CALL GEN$ADDR(BLC,LABELSTACK(MP));
      END;
      ELSE CALL ERROR('CE');
    END;
/* 124   <TRUE PART> ::= <BAL STMT>                                */
    DO;
      CALL GEN$ADDR(BRL,(LABELSTACK(SP-1)+1));
      CALL GEN$ADDR(LBL,LABELSTACK(SP-1));
    END;
/* 125   <LABEL DEF> ::= <LABEL> :                                 */
    IF LOOKUP$PN$ID(MP,LABL$ENTRY) THEN
    DO;
      CALL SETADDRPTR(5);
      CALL SETADDRPTR(6+BYTEPTR);
      CALL GEN$ADDR(LBL,ADDRPTR);
    END;
    ELSE CALL ERROR('UL');
/* 126   <SIMPLE STMT> ::= <ASSIGNMENT STMT>                       */
    ;
/* 127                      ^ <PROCEDURE STMT>                     */
    ;
/* 128                      ^ <WHILE STMT>                         */
    ;
/* 129                      ^ <REPEAT STMT>                        */
    ;
/* 130                      ^ <FOR STMT>                           */
    ;
/* 131                      ^ <CASE STMT>                          */
    ;
/* 132                      ^ <WITH STMT>                          */
    ;
```

```
/* 133                    ^ <GOTO STMT>                         */
    ;
/* 134                    ^ <COMPOUND STMT>                     */
    ;
/* 135                    ^                                     */
    ;
/* 136   <ASSIGNMENT STMT> ::= <VARIABLE> := <EXPRESSION> */
   CALL ASSIGN$VARI(MP,FALSE);
/* 137   <VARIABLE> ::= <VARIABLE IDENT>                        */
   ;
/* 138               ^ <VARIABLE> ^                             */
   ;
/* 139               ^ <VARIABLE> [ <EXPRES LIST> ]             */
   DO;
     TYPE$STACK(MP) = (TYPE$STACK(MP) OR 40H);
     TEMPBASE,BASE = BASE$LOC(MP);
     CALL SET$PAST$PN(9);
     BASE=ADDRPTR;
     CALL SETADDRPTR(5);
     IF BYTEPTR <> EXP$CTR THEN
       CALL ERROR ('XC');
     CALL SETADDRPTR(15);
     EXP$CTR1=EXP$CTR;
     DO WHILE EXP$CTR1 >0;
       CALL GEN$ADDR(LDII, ADDRPTR);
       APTRADDR = APTRADDR + 4;
       EXP$CTR1 = EXP$CTR1 - 1;
     END;
     CALL SETADDRPTR(11);
     CALL GEN$ADDR(LDII, ADDRPTR);
     BASE = TEMPBASE;
     CALL SET$PAST$PN(7);
     CALL GENADDR(LITA, ADDRPTR);
     CALL GENERATE(SUB);
     CALL GENERATE(EXP$CTR);
     CALL SETADDRPTR(9);
     BASE = ADDRPTR;
   END;
/* 140               ^ <VARIABLE> . <FIELD IDENT>               */
   IF NOT LOOKUP$ONLY(SP) THEN CALL ERROR('DT');
   ELSE DO;
     BASE = LOOKUP$ADDR;
     CALL SET$PAST$PN(12);
     PRT$ADDR(MP) = ADDRPTR + PRT$ADDR(MP);
     CALL SET$PAST$PN(9);
     CALL CASEPTRPTR(BYTEPTR);
     TYPE$STACK(MP), TYPE$STACK(SP) = PTRPTR;
   END;
/* 141   <VARIABLE IDENT> ::= <IDENTIFIER>                      */
   DO;
     VARPARM = FALSE;
     IF NOT LOOKUP$ONLY(SP) THEN
       CALL ERROR('DT');
     ELSE CALL SET$VAR$TYPE;/* LOOKUP$ADDR SET HERE */
```

```
    END;
/* 142   <EXPRES LIST> ::= <EXPRESSION>                          */
    EXP$CTR=1;
/* 143                      ^ <EXPRES LIST> , <EXPRESSION>      */
    EXP$CTR = EXP$CTR + 1;
/* 144   <EXPRESSION> ::= <SIMPLE EXPRESSION>                    */
    ;
/* 145                     ^ <SIMPLE EXPRESSION>                 */
/* 145                       <RELATIONAL OPERATOR>               */
/* 145                       <SIMPLE EXPRESSION>                 */
    IF CHK$EXPR$TYPE THEN CALL FIND$RELOP;
    ELSE CALL ERROR('CE');
/* 146   <RELATIONAL OPERATOR> ::= =                             */
    CALL SET$OP$TYPE(08H);
/* 147                               ^ < >                       */
    CALL SET$OP$TYPE(09H);
/* 148                               ^ < =                       */
    CALL SET$OP$TYPE(0AH);
/* 149                               ^ > =                       */
    CALL SET$OP$TYPE(0BH);
/* 150                               ^ <                         */
    CALL SET$OP$TYPE(0CH);
/* 151                               ^ >                         */
    CALL SET$OP$TYPE(0DH);
/* 152                               ^ IN                        */
    CALL SET$OP$TYPE(0EH);
/* 153   <TERM> ::= <FACTOR>                                     */
    ;
/* 154              ^ <TERM> <MULTIPLYING OPERATOR> <FACTOR>*/
    DO;
      IF READPARMS THEN
      DO;
        APTRADDR = PARMNUMLOC(MP);
        IF SHR(BYTEPTR,7) THEN
          CALL ERROR('NE');
      END;
      IF CHK$EXPR$TYPE THEN
      DO;
        DO CASE TYPE$STACK(MPP1);
  /*0*/ IF EXPRESS$STK(SP) = 1H THEN CALL GENERATE(MULI);
      ELSE IF EXPRESS$STK(SP) = 3H THEN CALL GENERATE(MULB);
      ELSE IF EXPRESS$STK(SP) = 0H THEN CALL GENERATE(ISEC);
             ELSE CALL ERROR('CE');
     /*1*/ IF EXPRESS$STK(SP) = 1H THEN
          DO;
            CALL GENERATE(CNVI);/* CONVERT 1ST INTEGER */
            CALL GENERATE(CN2I);/* CONVERT 2ND INTEGER */
            CALL GENERATE(DIVB);
            EXPRESS$STK(MP) = UNSIGN$EXPON;
          END;
      ELSE IF EXPRESS$STK(SP) = 3H THEN CALL GENERATE(DIVB);
          ELSE CALL ERROR('CE');
     /*2*/ IF EXPRESS$STK(SP)=INTEGER$TYPE THEN
                              CALL GENERATE(DIVI);
```

```
          ELSE CALL ERROR('CE');
      /*3*/ IF EXPRESS$STK(SP)=INTEGER$TYPE THEN
                                CALL GENERATE(MODX);
          ELSE CALL ERROR('CE');
      /*4*/ IF EXPRESS$STK(SP)=BOOLEAN$TYPE THEN
                                CALL GENERATE(ANDX);
          ELSE CALL ERROR('CE');
        END; /*  OF CASE VAR$TYPE$STK */
      END;
      ELSE CALL ERROR('CE');
    END;
/* 155   <MULTIPLYING OPERATOR> ::= *                          */
    CALL SET$OP$TYPE(00H);
/* 156                            ^ /                          */
    CALL SET$OP$TYPE(01H);
/* 157                            ^ DIV                        */
    CALL SET$OP$TYPE(02H);
/* 158                            ^ MOD                        */
    CALL SET$OP$TYPE(03H);
/* 159                            ^ AND                        */
    CALL SET$OP$TYPE(04H);
/* 160   <SIMPLE EXPRESSION> ::= <TERM>                        */
    ;
/* 161                         ^ <SIGN> <TERM>                 */
    DO;
      IF READPARMS THEN DO;
        APTRADDR = PARMNUMLOC(SP);
        IF SHR(BYTEPTR,7) THEN
          CALL ERROR('NE');
      END;
      IF SIGNTYPE = NEG THEN
      DO;
        IF EXPRESS$STK(SP) = UNSIGN$EXPON THEN
          CALL GENERATE(NEGB);
        ELSE IF EXPRESS$STK(SP) = INTEGER$TYPE THEN
          CALL GENERATE(NEGI);
        ELSE CALL ERROR('UO');
      END;
      SIGN$FLAG = FALSE;
      CALL COPY$STACKS(MP,SP);
    END;
/* 162                         ^ <SIMPLE EXPRESSION>           */
/* 162                           <ADDING OPERATOR> <TERM>      */
    DO;
      IF READPARMS THEN DO;
        APTRADDR = PARMNUMLOC(MP);
        IF SHR(BYTEPTR,7) THEN
          CALL ERROR('NE');
     END;
     IF CHK$EXPR$TYPE THEN
       DO;
         IF TYPE$STACK(MPP1)=5H THEN/* ARITH ADD */
         DO CASE EXPRESS$STK(SP);
           CALL GENERATE(UNION);  /* CASE 0 - ORD TYPE */
```

```
          CALL GENERATE(ADDI);  /* CASE 1 - INTEGER */
          CALL ERROR('CE');/* CASE 2 - CHAR */
          CALL GENERATE(ADDR);/* CASE 3 - REAL */
          CALL ERROR('CE');/* CASE 4 - STRING */
          CALL ERROR('CE');/* CASE 5 - BOOLEAN */
        END;/* CASE */
        ELSE IF TYPE$STACK(MPP1)= 6H THEN/* ARITE SUBTRC */
          DO CASE EXPRESS$STK(SP);
            CALL GENERATE(STDIF);/* CASE 0 - ORD TYPE */
            CALL GENERATE(SUBI);
            CALL ERROR('CE');
            CALL GENERATE(SUBR);
            CALL ERROR('CE');
            CALL ERROR('CE');
          END;
          ELSE IF TYPE$STACK(MPP1)=7H THEN/* BOOLEAN OR */
            DO;
              IF EXPRESS$STK(SP) = BOOLEAN$TYPE THEN
                CALL GENERATE(BOR);
              ELSE CALL ERROR('CE');
            END;
      END;
      ELSE CALL ERROR('CE');
     END;
/* 163   <ADDING OPERATOR> ::= +                          */
    CALL SET$OP$TYPE(05H);
/* 164                       ^ -                          */
    CALL SET$OP$TYPE(06H);
/* 165                       ^ OR                         */
    CALL SET$OP$TYPE(07H);
/* 166   <FACTOR> ::= <VARIABLE>                          */
    IF (FORM$FIELD(MP) = 05H) OR (FORM$FIELD(MP) = 0DH) THEN
      CALL CALL$A$PROC(FALSE);
    ELSE
      CALL LOAD$VARI(SP);
/* 167              ^ <VARIABLE> ( <ACTUAL PARA LIST> )   */
    CALL CALL$A$PROC(TRUE);
/* 168              ^ ( <EXPRESSION> )                    */
    CALL COPY$STACKS(MP, MPP1);
/* 169              ^ <SET>                               */
     ;
/* 170              ^ NOT <FACTOR>                        */
    DO;
      IF EXPRESS$STK(SP) = BOOLEAN$TYPE THEN
        CALL GENERATE(NOTX);
      ELSE CALL ERROR('CE');
      CALL COPY$STACKS(MP,SP);
    END;
/* 171              ^ <NUMBER>                            */
    IF TYPENUM=INTEGER$TYPE THEN
    DO;
      EXPRESS$STK(SP)  =INTEGER$TYPE;
      ALLC$QTY=CONVERTI(SP,POS);
      CALL GEN$ADDR(LDII,ALLC$QTY);
```

```
    END;
    ELSE DO;
      EXPRESS$STK(SP)  =UNSIGN$EXPON;
      CALL CONVRTECI(SP,POS);
      CALL GENERATE(LDIB);
      DO PTRPTR=0 TO BCDSIZE-1;
        CALL GENERATE(BCDNUM(PTRPTR));
      END;
    END;
/* 172        .        ^ NIL                                        */
    ;
/* 173                 ^ <STRING>                                   */
    DO;
      EXPRESS$STK(SP) = STRING$TYPE;
      CALL GENERATE(LDSI);
      DO FOREVER;
        DO PTRPTR = 1 TO ACCUM(0);
          CALL GENERATE(ACCUM(PTRPTR));
        END;
        IF CONT THEN/* STRING > 32 CHARS */
          CALL SCANNER;
        ELSE DO;
          CALL GENERATE(NOP);
          RETURN;
        END;
      END;
    END;
/* 174   <ACTUAL PARA LIST> ::= <ACTUAL PARA>                      */
      PARMNUM(SP) = 1;
/* 175                            ^ <ACTUAL PARA LIST> ,          */
/* 175                              <ACTUAL PARA>                  */
      PARMNUM(MP) = PARMNUM(MP) + 1;
/* 176   <SET> ::= [ <ELEMENT LIST> ]                              */
    CALL COPY$STACKS(MP, MPP1);
/* 177   <ELEMENT LIST> ::=                                        */
    CALL COPY$STACKS(SP, SP-3);
/* 178                       ^ <XELEMENT LIST>                     */
    ;
/* 179   <XELEMENT LIST> ::= <ELEMENT>                             */
    ;
/* 180                        ^ <XELEMENT LIST> , <ELEMENT>        */
    IF EXPRESS$STK(MP) <> EXPRESS$STK(SP) THEN
      CALL ERROR('ET');
/* 181   <ELEMENT> ::= <EXPRESSION>                                */
    ;
/* 182                  ^ <EXPRESSION> .. <EXPRESSION>            */
IF EXPRESS$STK(MP) <> EXPRESS$STK(SP) THEN CALL ERROR('ET');
/* 183   <GOTO STMT> ::= GOTO <LABEL>                              */
    IF LOOKUP$PN$IL(SP,LABEL$ENTRY) THEN
    DO;
      CALL SETADDRPTR(5);
      CALL SETADDRPTR(6+BYTEPTR);
      CALL GEN$ADDR(BRL,ADDRPTR);
    END;
```

```
    ELSE DO;
      CALL ERROR('UL');
      CALL GENERATE(NOP); CALL GENERATE(NOP);
    END;
/* 184   <COMPOUND STMT> ::= BEGIN <STMT LISTS>  END        */
    ;
/* 185   <STMT LISTS> ::= <STMT>                             */
    ;
/* 186                  ^ <STMT LISTS> ; <STMT>              */
    ;
/* 187   <PROCEDURE STMT> ::= <PROCEDURE IDENT>              */
    CALL CALL$A$PROC(FALSE);
/* 188                      ^ <PROCEDURE IDENT> (            */
/* 188                        <ACTUAL PARA LIST> )           */
    IF FORM$FIELD(MP) = BUILT$IN$PROC THEN
      CALL CALL$A$PROC(FALSE);
    ELSE CALL CALL$A$PROC(TRUE);
/* 189   <PROCEDURE IDENT> ::= <IDENTIFIER>                  */
    DO;
      IF NOT LOOKUP$ONLY(SP) THEN
        CALL ERROR('UP');
      ELSE DO;
        BASELOC(SP) = LOOKUP$ADDR;
        CALL SETADDRPTR(4);
        FORM$FIELD(SP) = BYTEPTR;
        IF FORM$FIELD(SP) = BUILT$IN$PROC THEN
        DO;
          CALL SET$PAST$PN(7);
          IF BYTEPTR = 28 THEN
          DO;
            PARMNUM(SP) = 2;
            PARMNUMLOC(SP) = APTRADDR + 1;
          END;
          ELSE IF BYTEPTR > 21 THEN
            DO CASE (BYTEPTR - 22);
              NEW$STMT = TRUE;
              DISPOSE$STMT = TRUE;
              REAL$STMT = TRUE;
              READ$STMT = TRUE;
              WRITE$STMT = TRUE;
              WRITE$STMT = TRUE;
            END;    /* OF CASE (BYTEPTR - 22) */
        END;
        ELSE DO;/* NOT BUILT IN */
          CALL SET$PAST$PN(7);
          PARMNUM(SP) = BYTEPTR;
          CALL SET$PAST$PN(8);
          PARMNUMLOC(SP) = ADDRPTR;
          APTRADDR = APTRADDR + 6;
          LABELSTACK(SP) = ADDRPTR;
          READPARMS = TRUE;
          PARMNUMLOC(SP+2) = PARMNUMLOC(SP);
        END;
      END;
```

```
    END;
/* 190   <ACTUAL PARA> ::= <EXPRESSION>                          */
    IF READ$STMT THEN CALL READ$VAR;
    ELSE IF WRITE$STMT THEN CALL WRITE$VAR(0);
      ELSE IF NOT(READPARMS) THEN
      DO:
        READPARMS = TRUE;
        CALL GENERATE(PARMX);
             /*PARAMETER IS AN EXPRESSION VALUE */
      END;
/* 191                  ^ <EXPRESSION> : <EXPRESSION>        */
    IF NOT WRITE$STMT THEN CALL ERROR('PE');
    ELSE DO;
   IF EXPRESS$STK(SP) <> INTEGER$TYPE THEN CALL ERROR('WP');
      CALL WRITE$VAR(1);
    END;
/* 192                  ^ <EXPRESSION> : <EXPRESSION> :   */
/* 192                    <EXPRESSION>                      */
    IF NOT WRITE$STMT THEN CALL ERROR('PE');
    ELSE DO;
   IF EXPRESS$STK(MP) <> UNSIGN$EXPON THEN CALL ERROR('RT');
      IF (EXPRESS$STK(SP) <> INTEGER$TYPE) AND
  (EXPRESS$STK(SP-2) <> INTEGER$TYPE) THEN CALL ERROR('WP');
      CALL WRITE$VAR(2);
    END;
/* 193 <CASE STMT> ::=<CASE EXPRESS><CASE LIST ELEMT LIST>*/
/* 193                      END                              */
    DO;
      LABLCOUNT = LABLCOUNT + 1;
      CALL GEN$ADDR(LBL,LABELSTACK(MP)):
      CASE$COUNT = CASE$COUNT - 1;
    END;
/* 194   <CASE EXPRESS> ::= CASE <EXPRESSION> OF              */
    DO:
     CASE$STMT=TRUE;
     IF (EXPRESS$STK(MPP1) = UNSIGN$EXPON) THEN
                              CALL ERROR('RT');
     LABELSTACK(MP) = LABLCOUNT;
     LABLCOUNT = LABLCOUNT + 1;
     CASE$STK(CASE$COUNT := CASE$COUNT + 1) = 0;
    END;
/* 195   <CASE LIST ELEMT LIST> ::= <CASE LIST ELEMENT>   */
    IF CASE$STMT THEN
    DO;
      CALL GEN$ADDR(BRL,LABELSTACK(MP-1));
      CALL GEN$ADDR(LBL,(LABELSTACK(MP)+1));
    END;
/* 196                        ^ <CASE LIST ELEMT LIST> ; */
/* 196                            <CASE LIST ELEMENT>   */
    IF CASE$STMT THEN
    DO;
      CALL GEN$ADDR(BRL,LABELSTACK(MP-1));
      CALL GEN$ADDR(LBL,(LABELSTACK(SP)+1));
    END;
```

```
/* 197   <CASE LIST ELEMENT> ::=                                */
    CASE$STMT = FALSE;
/* 198                              ^ <CASE PREFIX> <STMT>      */
    ;
/* 199   <CASE PREFIX> ::= <CASE LABEL LIST> :                  */
    DO;
      CALL GEN$ADDR(BRL,(LABELSTACK(MP)+1));
      CALL GEN$ADDR(LBL,LABELSTACK(MP));
    END;
/* 200   <WITH STMT> ::= <WITH> <REC VARIABLE LIST> <DO>   */
/* 200                    <BAL STMT>                            */
    ;
/* 201   <WITH> ::= WITH                                        */
    ;
/* 202   <REC VARIABLE LIST> ::= <VARIABLE>                     */
    ;
/* 203                              ^ <REC VARIABLE LIST> ,     */
/* 203                                <VARIABLE>                */
    ;
/* 204   <DO> ::= DO                                            */
    DO;
      LABELSTACK(SP) = LABLCOUNT;
      CALL GEN$ADDR(BLC,LABELSTACK(SP));
      LABLCOUNT = LABLCOUNT + 1;
    END;
/* 205  <WHILE STMT> ::=<WHILE><EXPRESSION><DO><BAL STMT> */
    DO;
      CALL GEN$ADDR(BRL,LABELSTACK(MP));
      CALL GEN$ADDR(LBL,LABELSTACK(SP-1));
    END;
/* 206   <WHILE> ::= WHILE                                      */
    DO;
      LABELSTACK(SP) = LABLCOUNT;
      CALL GEN$ADDR(LBL,LABELSTACK(SP));
      LABLCOUNT = LABLCOUNT + 1;
    END;
/* 207  <FOR STMT> ::= FOR<CONTROL VARIABLE> :=<FOR LIST> */
/* 207                  <DO> <BAL STMT>                         */
    DO;
      CALL GEN$ADDR(BRL,(LABELSTACK(SP-2)+1));
      CALL GEN$ADDR(LBL,LABELSTACK(SP-1));
    END;
/* 208   <FOR LIST> ::= <INITIAL VALUE> <TO> <FINAL VALUE>*/
    DO;
      IF EXPRESS$STK(MP) <> EXPRESS$STK(SP) THEN
                              CALL ERROR('ET');
      CALL GENERATE(GEQI);
    END;
/* 209              ^ <INITIAL VALUE> <DOWNTO> <FINAL VALUE> */
    DO;
      IF EXPRESS$STK(MP) <> EXPRESS$STK(SP) THEN
                              CALL ERROR('ET');
      CALL GENERATE(LEQI);
    END;
```

```
/* 210   <CONTROL VARIABLE> ::= <IDENTIFIER>                          */
    DO;
      VARPARM = FALSE;
      IF NOT LOOKUP$ONLY(SP) THEN
        CALL ERROR('CV');
      ELSE DO;
        APTRADDR = LOOKUP$ADDR + 4;
        IF BYTEPTR = 1BH THEN CALL ERROR('CV');
        ELSE CALL SET$VAR$TYPE;
      END;
    END;
/* 211   <INITIAL VALUE> ::= <EXPRESSION>                             */
    DO;
      CALL ASSIGN$VARI(SP-2, TRUE);
      LABELSTACK(SP) = LABLCOUNT;
      LABLCOUNT = LABLCOUNT + 2;
      CALL GEN$ADDR(BRL,LABELSTACK(SP));
      CALL GEN$ADDR(LBL,(LABELSTACK(SP)+1));
      CALL LOAD$VARI(SP-2);
    END;
/* 212   <FINAL VALUE> ::= <EXPRESSION>                               */
    ;
/* 213   <REPEAT STMT> ::= <REPEAT> <STMT LISTS> UNTIL                */
/* 213                     <EXPRESSION>                               */
    DO;
      IF EXPRESS$STK(SP) = BOOLEAN$TYPE THEN
      DO;
        CALL GENERATE(NOTX);
        CALL GEN$ADDR(BLC,LABELSTACK(MP));
      END;
      ELSE CALL ERROR('CE');
    END;
/* 214   <REPEAT> ::= REPEAT                                          */
    DO;
      CALL GEN$ADDR(LBL,LABLCOUNT);
      LABELSTACK(SP)=LABLCOUNT;
      LABLCOUNT = LABLCOUNT + 1;
    END;
/* 215   <TO> ::= TO                                                  */
    DO;
      CALL GENERATE(INC);
      CALL GEN$ADDR(LBL,LABELSTACK(SP-1));
    END;
/* 216   <DOWNTO> ::= DOWNTO                                          */
    DO;
      CALL GENERATE(DEC);
      CALL GEN$ADDR(LBL,LABELSTACK(SP-1));
    END;
END; /* OF CASE STATEMENT */

 END SYNTHESIZE;


 END SYNTH2;
```

```
DECODE:DO;
DECLARE
LIT             LITERALLY      'LITERALLY',
FCB             ADDRESS        INITIAL(5CH),
FCB$BYTE        BASED          FCB (1) BYTE,
I               BYTE,
F0H             BYTE           INITIAL(0F0H),
DECI(3)         BYTE           INITIAL(64H,0AH,1H),
TRUE            LIT            '1',
FALSE           LIT            '0',
ADDR            ADDRESS        INITIAL(100H),
CHAR            BASED          ADDR BYTE,
DCL             LIT            'DECLARE',
EXT             LIT            'EXTERNAL',
PROC            LIT            'PROCEDURE',
BUFF$END        LIT            '0FFH',
BCDNUM(8)       BYTE;


MON1:PROC(FUNC,INFO) EXT;
     DCL FUNC BYTE,
         INFO ADDRESS;
END MON1;


MON2: PROC(FUNC,INFO) BYTE EXT;
     DCL FUNC BYTE,
         INFO ADDRESS;
END MON2;


  BOOT: PROC EXT;
  END BOOT;


PRINT$CHAR: PROCEDURE (CHAR);
    DECLARE CHAR BYTE;
    CALL MON1(2,CHAR);
 END PRINT$CHAR;


 CRLF: PROC;
    CALL PRINT$CHAR(13);
    CALL PRINT$CHAR(10);
END CRLF;
```

```
P: PROCEDURE(ADD1);
    DECLARE ADD1 ADDRESS, C BASED ADD1 (1) BYTE;
    CALL CRLF;
    DO I=0 TO 4;
         CALL PRINT$CHAR(C(I));
    END;
    CALL PRINT$CHAR(' ');
  END P;


GET$CHAR: PROCEDURE BYTE;
    IF (ADDR:=ADDR+1) > BUFF$END THEN
    DO;
         IF MON2(20,FCB) <> 0 THEN
         DO;
              CALL P(.('END  '));
         END;
         ADDR=80H;
    END;
    RETURN CHAR;
END GET$CHAR;


WRITE$STRING: PROCEDURE;
    DECLARE J BYTE;
    DO WHILE 1;
      J = GET$CHAR;
      IF J <> 00H THEN CALL PRINT$CHAR(J);
      ELSE RETURN;
    END;
END WRITE$STRING;


D$CHAR: PROCEDURE(OUTPUT$BYTE);
    DECLARE OUTPUT$BYTE BYTE;
    IF OUTPUT$BYTE < 10 THEN CALL PRINT$CHAR(OUTPUT$BYTE +
                                                   30H);
    ELSE CALL PRINT$CHAR(OUTPUT$BYTE + 37H);
END D$CHAR;


D: PROCEDURE (COUNT);
    DECLARE (COUNT, J) ADDRESS;
    DO J=1 TO COUNT;
         CALL D$CHAR(SHR(GET$CHAR,4));
         CALL D$CHAR(CHAR AND 0FH);
         CALL PRINT$CHAR(' ');
    END;
END D;
```

```
PRINT$BCD: PROCEDURE (COUNT);
DCL (COUNT,J,L,K) BYTE;


P$EXPON: PROCEDURE(VALUE);
  DCL (VALUE,X,COUNT1) BYTE;
  DCL       FLAG       BYTE;

  DO X = Ø TO 2;
  FLAG = FALSE;
     COUNT1 = 3ØH;
     DO WHILE VALUE >= DECI(X);
        VALUE = VALUE - DECI(X);
        FLAG = TRUE;
        COUNT1 = COUNT1 +1;
     END;
     IF FLAG OR (X >= 2) THEN
           CALL PRINT$CHAR(COUNT1);
           ELSE CALL PRINT$CHAR(' ');
 END;
 RETURN;
END P$EXPON;


    DO L = Ø TO (COUNT-1);
      BCDNUM(L) = GET$CHAR;
    END;
    CALL PRINT$CHAR(' ');
 IF BCDNUM(COUNT-1) >= 8ØH THEN CALL PRINT$CHAR('-');
    ELSE CALL PRINT$CHAR('+');
 CALL PRINT$CHAR('Ø');
 CALL PRINT$CHAR('.');
DO L=Ø TO COUNT-2;
 J,K = BCDNUM(L);
 K = SHR((K AND FØH),4); /* EXTRACT THE MSD FM THE BYTE */
*****************************************************************
 CALL D$CHAR(K);
 J = (J AND ØFH);    /* EXTRACT THE LSD FM THE BYTE */
 CALL D$CHAR(J);
 END;
 J,K = (BCDNUM(COUNT-1) AND 7FH); /* GET RID OF SIGN */
 IF K >=4ØH THEN CALL PRINT$CHAR('+');
    ELSE CALL PRINT$CHAR('-'); /* SIGN OF EXPONENT */
 CALL PRINT$CHAR('E');
CALL P$EXPON(K AND 3FH);
END PRINT$BCD;



PRINT$REST: PROCEDURE;
    DECLARE
```

```
    ENDP          LIT        '1H',
    FOLS          LIT        '23H',
    NEGS          LIT        '24H',
    LBL           LIT        '2H',
    LDII          LIT        '4H',
    ALL           LIT        '0BH',
    LITA          LIT        '0CH',
    BRL           LIT        '3FH',
    BLC           LIT        '3FH',
    PRO           LIT        '5H',
    ANDX          LIT        '32H',
    BOR           LIT        '33H',
    PARM          LIT        '43H',
    PARMV         LIT        '44H',
    LDIB          LIT        '3H',
    WRTB          LIT        '53H',
    WRTI          LIT        '54H',
    WRTS          LIT        '55H',
    LDSI          LIT        '4BH',
    KASE           LIT        '4CH';
    IF CHAR = ENDP THEN
    DO;
         CALL P(.('END  '));
         CALL BOOT;
    END;
    IF   (CHAR=WRTB) OR (CHAR=WRTI) OR
         (CHAR=WRTS) THEN DO; CALL D(1); RETURN; END;
IF (CHAR=LBL) OR (CHAR=LDII) OR (CHAR=ALL) OR (CHAR=LITA) OR
    (CHAR=BRL) OR (CHAR=BLC) OR (CHAR=PRO) OR
           (CHAR=PARM) OR (CHAR=PARMV) THEN DO;
         CALL D(2); RETURN; END;
    IF CHAR = KASE THEN DO; CALL D(4); RETURN; END;
    IF CHAR = LDIB THEN DO; CALL PRINT$BCD(8); RETURN; END;
    IF CHAR = LDSI THEN DO; CALL WRITE$STRING; RETURN; END;
    RETURN;
END PRINT$REST;




/*****   PROGRAM EXECUTION STARTS HERE   *****/
MAINLINE: DO;
IF MON2(15,FCB) = 255 THEN
DO;
    CALL P(.('NO FILE FOUND'));
    CALL BOOT;
END;
DO WHILE 1;
IF GET$CHAR <= 72H THEN
   DO CASE CHAR;
         CALL P(.('NOP  '));
```

```
CALL P(.('ENDP '));
CALL P(.('LBL  '));
CALL P(.('LDIB '));
CALL P(.('LDII '));
CALL P(.('PRO  '));
CALL P(.('RTN  '));
CALL P(.('SAVP '));
CALL P(.('UNSP '));
CALL P(.('CNVB '));
CALL P(.('CNVI '));
CALL P(.('ALL  '));
CALL P(.('LITA '));
CALL P(.('ADDB '));
CALL P(.('ADDI '));
CALL P(.('SUBB '));
CALL P(.('SUBI '));
CALL P(.('MULB '));
CALL P(.('MULI '));
CALL P(.('DIVB '));
CALL P(.('DIVI '));
CALL P(.('MODX '));
CALL P(.('EQLI '));
CALL P(.('NEQI '));
CALL P(.('LEQI '));
CALL P(.('GEQI '));
CALL P(.('LSSI '));
CALL P(.('GTRI '));
CALL P(.('XIN  '));
CALL P(.('EQLB '));
CALL P(.('NEQB '));
CALL P(.('LEQB '));
CALL P(.('GEQB '));
CALL P(.('LSSB '));
CALL P(.('GRTB '));
CALL P(.('EQLS '));
CALL P(.('NEQS '));
CALL P(.('LEQS '));
CALL P(.('GEQS '));
CALL P(.('LSSS '));
CALL P(.('GRTS '));
CALL P(.('EQSET'));
CALL P(.('NEQST'));
CALL P(.('INCL1'));
CALL P(.('INCL2'));
CALL P(.('NEGB '));
CALL P(.('NEGI '));
CALL P(.('COMB '));
CALL P(.('COMI '));
CALL P(.('NOTX '));
CALL P(.('ANDX '));
CALL P(.('BOR  '));
CALL P(.('STOB '));
CALL P(.('STOI '));
CALL P(.('STO  '));
```

```
CALL P(.('STDB '));
CALL P(.('STDI '));
CALL P(.('STD  '));
CALL P(.('UNION'));
CALL P(.('STDIF'));
CALL P(.('ISFC '));
CALL P(.('CNAI '));
CALL P(.('BRL  '));
CALL P(.('BLC  '));
CALL P(.('CN2I '));
CALL P(.('MKSET'));
CALL P(.('XCHG '));
CALL P(.('PARM '));
CALL P(.('PARMV'));
CALL P(.('PARMX'));
CALL P(.('INC  '));
CALL P(.('DEC  '));
CALL P(.('DEL  '));
CALL P(.('WRT  '));
CALL P(.('SUB  '));
CALL P(.('LDSI '));
CALL P(.('CASE '));
CALL P(.('LOD  '));
CALL P(.('LODB '));
CALL P(.('LODI '));
CALL P(.('RDVB '));
CALL P(.('RDVI '));
CALL P(.('RDVS '));
CALL P(.('WRTB '));
CALL P(.('WRTI '));
CALL P(.('WRTS '));
CALL P(.('DUMP '));
CALL P(.('ABS  '));
CALL P(.('SQR  '));
CALL P(.('SIN  '));
CALL P(.('COS  '));
CALL P(.('ARCTN'));
CALL P(.('EXP  '));
CALL P(.('LN   '));
CALL P(.('SQRT '));
CALL P(.('ODD  '));
CALL P(.('EOLN '));
CALL P(.('EOF  '));
CALL P(.('TRUNC'));
CALL P(.('ROUND'));
CALL P(.('ORD  '));
CALL P(.('CHR  '));
CALL P(.('SUCC '));
CALL P(.('PRED '));
CALL P(.('SEEK '));
CALL P(.('PUT  '));
CALL P(.('GET  '));
CALL P(.('RESET'));
CALL P(.('REWRT'));
```

```
          CALL P(.('PAGE '));
          CALL P(.('NEW  '));
          CALL P(.('IISPZ'));
          CALL P(.('FWD  '));
          CALL P(.('XTRNL'));
          CALL P(.('RIV  '));
     END;                            /* OF CASE STATEMENT */
     ELSE CALL P(.('ZZZZ '));
     CALL PRINT$REST;
   END;    /* OF DO WHILE */;
END MAINLINE;
END DECODE;
```

```
$PAGEWIDTH(80) TITLE('SYM - SYMBOL TABLE PRINT UTILITY')
 /*

-LINK SYM.OBJ,TRINT.OBJ,PLM80.LIB TO SYM.LNK

-LOCATE SYM.LNK CODE(103H)

 **************************************************************
 *                                                            *
 *                                                            *
 *                    SYMBOL$TABLE$PRINTOUT                   *
 *                                                            *
 *                                                            *
 **************************************************************


  THIS PROGRAM TAKES THE OUTPUT FROM THE PASCAL SYMBOL TABLE
    AND CONVERTS IT INTO A READABLE OUTPUT TO FACILITATE
    DEBUGGING.
    */


SYM:DO;

DECLARE

LIT             LITERALLY       'LITERALLY',
EXT             LIT              'EXTERNAL',
FCB             ADDRESS         INITIAL (5CH),
ADDR            LIT             'ADDRESS',
FCB$BYTE        BASED           FCB (1) BYTE,
DECI5(5)       ADDR              INITIAL(10000,1000.100,10,1),
DECI(3)         BYTE             INITIAL(100,10,1),
I               BYTE,
TRUE            LIT             '1',
FALSE           LIT             '0',
COPYING         BYTE            INITIAL(TRUE),
ADDR1           ADDRESS         INITIAL (100H),
CHAR            BASED           ADDR1 BYTE,
BUFF$END        LIT             '0FFH',
FORMMASK        LIT             '07H',
DCL             LIT             'DECLARE';
DCL
PROC            LIT             'PROCEDURE',
EOFFILLER       LIT             '1AH',
BCDNUM(8)       BYTE,
FILE$TYPE(3)      BYTE                DATA('S','Y','M'),
FORM            BYTE,
TABLE$START     ADDR, /* STARTING LOCATION AT COMPILATION */
OFFSET          ADDR, /* NEW VALUE OF TABLE ENTRY */
```

```
PARM$LISTING(10)  ADDR, /* LOCATION OF SUBPTN FORMAL PARAM
                                                    LISTING */
SUBPTN  BYTE INITIAL(0),
PARM$NUM(10) BYTE, /* KEEPS COUNT OF NUMBER OF PARAMETERS */
SAVE$BASE      ADDR, /* SAVES BASE LOCATION */
LPN            BYTE , /* LENGTH OF PRINTNAME */

BASE                    ADDR,/*BASE OF CURRENT ENTRY*/
SBTBLTOP                ADDR, /* CURRENT TOP OF TABLE(SYM)*/
SBTBL                   ADDR,
L                       ADDR,/*LENGTH OF SYMBOL TABLE*/
PTR                     BASED BASE BYTE,/*1ST BYTE OF ENTRY*/
APTRADDR                ADDR,/* UTILITY VAR FOR TABLE*/
ADDRPTR                 BASED APTRADDR ADDR,
BYTEPTR                 BASED APTRADDR BYTE,
PRINTNAME              ADDR,/* SET PRIOR TO LOOKUP OR ENTER */
SYMHASH                 BYTE,
        LAST$SBTBL$ID ADDR,
        PARAMNUMLOC   ADDR,
        SBTBLSCOPE    ADDR;


MON1: PROCEDURE (F,A) EXT;
    DECLARE F BYTE, A ADDRESS;
END MON1;


MON2: PROCEDURE (F,A) BYTE EXT;
    DECLARE F BYTE, A ADDRESS;
END MON2:


BOOT: PROC EXT:
END BOOT;



PRINT$CHAR: PROCEDURE (CHAR);
    DECLARE CHAR BYTE;
    CALL MON1(2,CHAR);
END PRINT$CHAR;


CRLF: PROCEDURE;
    CALL PRINT$CHAR(13);
    CALL PRINT$CHAR(10);
END CRLF;


PRINT: PROC(A);
    DCL      A ADDR;
    CALL MON1(9,A);
END PRINT;
```

```
GET$CHAR: PROCEDURE BYTE;
    IF (ADDR1:=ADDR1+1) > BUFF$END THEN
    DO;
         IF MON2(20,FCB) <> 0 THEN
         DO;
               CALL PRINT(.('THE END  $'));
         END;
         ADDR1=80H;
    END;
    RETURN CHAR;
END GET$CHAR;


D$CHAR: PROCEDURE(OUTPUT$BYTE);
    DECLARE OUTPUT$BYTE BYTE;
    IF OUTPUT$BYTE < 10 THEN CALL PRINT$CHAR(OUTPUT$BYTE +
                                                         30H);
    ELSE CALL PRINT$CHAR(OUTPUT$BYTE + 37H);
END D$CHAR;


D: PROCEDURE (COUNT);
    DECLARE (COUNT, J) ADDRESS;
    DO J=1 TO COUNT;
         CALL D$CHAR(SHR(BYTEPTR,4));
         CALL D$CHAR(BYTEPTR AND 0FH);
         APTRADDR = APTRADDR + 1;
    END;
END D;


/***************************************************************/

PRINT$BCD: PROCEDURE (COUNT);
    DECLARE (COUNT, J, K, L) BYTE;

   P$EXPON: PROCEDURE(VALU);
      DECLARE (VALU, X, COUNT1) BYTE;
      DECLARE FLAG BYTE;
      DO X = 0 TO 2;
      FLAG = FALSE;
        COUNT1 = 30H;
        DO WHILE VALU >= DECI(X);
          VALU = VALU - DECI(X);
          FLAG = TRUE;
          COUNT1 = COUNT1 + 1;
        END;
        IF FLAG OR (X >= 2) THEN
          CALL PRINT$CHAR(COUNT1);
        ELSE CALL PRINT$CHAR(' ');
      END;
      RETURN;
```

```
   END P$EXPON;


DO L= 0 TO (COUNT-1);
  BCDNUM(L) = BYTEPTR;
  APTRADDR=APTRADDR+1;
END;
CALL PRINT$CHAR(' ');
IF BCDNUM(COUNT-1) >= 80H THEN CALL PRINT$CHAR('-');
   ELSE CALL PRINT$CHAR('+');
CALL PRINT$CHAR('0');
CALL PRINT$CHAR('.');
DO L=0 TO COUNT-2;
J,K=BCDNUM(L);
K= SHR((K AND 0F0H),4); /* EXTRACT THE MSD FROM THE BYTE */
CALL D$CHAR(K);
J = (J AND 0FH);        /* EXTRACT THE LSD FROM THE BYTE */
CALL D$CHAR(J);
END;
J,K = (BCDNUM(COUNT-1) AND 7FH); /* GET RID OF  SIGN */
IF K >= 40H THEN CALL PRINT$CHAR('+');
 ELSE CALL PRINT$CHAR('-');   /* SIGN OF EXPONET  */
CALL PRINT$CHAR('E');
CALL P$EXPON(K AND 3FH);
END PRINT$BCD;



DOTSYM:PROCEDURE;

FCB$BYTE(32), FCB$BYTE(0) = 0;
DO I = 0 TO 2;
    FCB$BYTE(I+9) = FILE$TYPE(I);
END;

IF MON2(15,FCB) = 255 THEN
DO;
    CALL PRINT(.('ERROR--GONE TO BOOT $'));
    CALL BOOT;
END;

END DOTSYM;


DISKERR: PROC;
    DO;
       CALL PRINT(.('DE    $'));
       CALL BOOT;
    END;
END DISKERR;


PRINTDEC: PROC(VALUE);
    DCL     VALUE ADDR, I BYTE, COUNT BYTE;
```

```
    DCL      FLAG BYTE;
    FLAG = FALSE;
    DO I = 0 TO 4;
    COUNT = 30H;
       DO WHILE VALUE >= DECI5(I);
          VALUE = VALUE - DECI5(I);
          FLAG= TRUE;
          COUNT = COUNT + 1;
       END;
       IF FLAG OR (I>= 4) THEN
          CALL PRINTCHAR(COUNT);
       ELSE
          CALL PRINTCHAR(' ');
    END;
    RETURN;
END PRINTDEC;


SETADDRPTR: PROC(OFFSET);
    DCL OFFSET ADDR;
    APTRADDR = BASE + OFFSET;
END SETADDRPTR;



SET$PAST$PN: PROC(OFFSET);
    DCL OFFSET BYTE;
    CALL SETADDRPTR(6);
    CALL SETADDRPTR(BYTEPTR + OFFSET);
END SET$PAST$PN;


COPY$SBTBL: PROC ADDR;
/* COPIES FILE.SYM TO MEMORY, LOOKING FOR TWO EOFFILLERS
   (1AH) IN A ROW */
    DCL K ADDR;
    K = 0;
    DO WHILE COPYING;
          CALL SETADDRPTR(K);
          BYTEPTR = GETCHAR;
          K = K + 1;
          IF BYTEPTR = EOFFILLER THEN
          DO;
                K = K + 1;
                CALL SETADDRPTR(K);
                BYTEPTR = GETCHAR;
                IF BYTEPTR = EOFFILLER THEN
                DO;
                  COPYING = FALSE;
                  BYTEPTR = 00H;
                END;
          END;
    END;
    RETURN K;
```

```
END COPY$SBTBL;



RESET$LOCATION: PROC(A) ADDR;
    DCL A ADDR;
    OFFSET = A - TABLE$START;
    RETURN OFFSET;
END RESET$LOCATION;


TAB1: PROC;
    CALL PRINT(.('     $'));
END TAB1;


TAB2: PROC;
    CALL TAB1;
    CALL TAB1;
END TAB2;


WRITE$ENTRY: PROC;
    DO CASE (FORM AND 07H);
         CALL PRINT(.('LABEL ENTRY $'));
         CALL PRINT(.('CONSTANT ENTRY $'));
         CALL PRINT(.('TYPE ENTRY $'));
         CALL PRINT(.('VARIABLE ENTRY $'));
         CALL PRINT(.('PROCEDURE ENTRY $'));
         CALL PRINT(.('FUNCTION ENTRY $'));
         CALL PRINT(.('FILE ENTRY $'));
         CALL PRINT(.('USER DECLARED ENTRY $'));
    END;  /* CASE */
END WRITE$ENTRY;


PRINT$ID: PROC;
    DCL SIZE BYTE;
    CALL SETADDRPTR(6);
    SIZE = BYTEPTR;
    DO I = 1 TO SIZE;
         CALL SETADDRPTR(6+I);
         CALL PRINT$CHAR(BYTEPTR);
    END;
    CALL CRLF;
END PRINT$ID;


RANGER: PROC(A);
    DCL (A, BASE1) ADDR;
    BASE1 = BASE;
    BASE = A;
    CALL SET$ADDR$PTR(7);
    CALL CRLF;
```

```
    CALL TAB2;
    CALL PRINT(.('WITH LOW VALUE $'));
    IF (SHR(FORM,7) AND FORMMASK) THEN
      CALL PRINT$CHAR(BYTEPTR);
    ELSE CALL PRINT$DEC(ADDRPTR);
    CALL PRINT(.(' AND HIGH VALUE $'));
    CALL SET$ADDR$PTR(9);
    IF (SHR(FORM,7) AND FORMMASK) THEN
      CALL PRINT$CHAR(BYTEPTR);
    ELSE CALL PRINT$DEC(ADDRPTR);
    BASE = BASE1;
END RANGER;


USER$DEFINED: PROC;
     DO CASE (SHR(BYTEPTR,3) AND FORMMASK);
      DO;
        CALL PRINT(.('ENUMERATED TYPE - $'));
        CALL PRINT$ID;
        CALL PRINT(.('THE VALUE IS $'));
        CALL SET$PAST$PN(7);
        CALL PRINT$DEC(BYTEPTR);
      END;
      DO;
        DO CASE (SHR(BYTEPTR,6) AND FORMMASK);
          CALL PRINT(.('AN ENUMERATED SUBRANGE $'));
          DO;
            CALL PRINT(.('AN INTEGER SUBRANGE $'));
            LPN = LPN + 13;  /* LENGTH OF 4FH ENTRY */
          END;
          CALL PRINT(.('A CHARACTER SUBRANGE $'));
        END; /* OF CASE */
        CALL RANGER(BASE);
      END;
      DO;
        CALL PRINT(.('AN ARRAY $'));
        CALL SETADDRPTR(5);
        I = BYTEPTR;
        LPN = LPN + 13 + (4*I);  /* LENGTH OF 17H ENTRY */
        CALL CRLF;
        CALL TAB2;
        CALL PRINT(.('THE COMPONENT TYPE IS $'));
        CALL SETADDRPTR(10);
        DO CASE BYTEPTR;
          CALL PRINT(.('SCALAR $'));
          CALL PRINT(.('INTEGER $'));
          CALL PRINT(.('CHAR $'));
          CALL PRINT(.('REAL $'));
          CALL PRINT(.('STRING $'));
          CALL PRINT(.('BOOLEAN $'));
        END; /* OF CASE */
        CALL CRLF;
        CALL TAB2;
        CALL PRINT(.(' IT REQUIRES $'));
```

```
        CALL SETADDRPTR(8);
        CALL PRINT$DEC(ADDRPTR);
        CALL PRINT(.(' BYTES OF STORAGE$'));
        CALL CRLF;
        CALL TAB2;
        CALL PRINT(.('THERE IS/ARE $'));
        CALL PRINT$DEC(I);
        CALL PRINT(.(' DIMENSIONS IN THIS ARRAY $'));
        CALL SETADDRPTR(9);
        DO WHILE I <> 0;
          APTRADDR = APTRADDR + 4;
          CALL RANGER(ADDRPTR);
          LPN = LPN +13;  /* LENGTH OF 4TH ENTRY */
          I = I - 1;
        END;
      END;
      DO;
      END;
      DO;
        CALL PRINT(.('A SET OF $'));
        CALL SETADDRPTR(5);
        SAVEBASE = BASE;
        BASE = ADDRPTR;
        CALL PRINT$ID;
        BASE = SAVE$BASE;
      END;
      DO;
        CALL PRINT(.('A FILE OF $'));
        CALL SETADDRPTR(5);
        SAVEBASE = BASE;
        BASE = ADDRPTR;
        CALL PRINT$ID;
        BASE = SAVEBASE;
      END;
      DO;
        CALL PRINT(.('A POINTER OF TYPE $'));
        CALL SETADDRPTR(5);
        SAVE$BASE = BASE;
        BASE = ADDRPTR;
        CALL PRINT$ID;
        BASE = SAVE$BASE;
      END;
    END; /* OF CASE */
END USER$DEFINED;


CHECK$COLLISION: PROC;
/* LOOKS FOR ADDRESS IN COLLISION FIELD, THEN READS
   COLLISION CHAIN BACKWARD, PRINTING PRINTNAMES. STOPS
   WHEN NO FURTHUR COLLISIONS OR TABLE RUNS OUT. */
    CALL SETADDRPTR(6);
    LPN = BYTEPTR;
    CALL TAB1;
    CALL PRINT(.('HASH VALUE = $'));
```

```
    CALL SETADDRPTR(5);
    CALL PRINT$DEC(BYTEPTR);
    CALL SETADDRPTR(3);
    IF ADDRPTR = 00H THEN
     CALL PRINT(.(' AND THERE ARE NO COLLISIONS $'));
    ELSE DO;
         SAVE$BASE = BASE;
         DO WHILE ADDRPTR >= TABLE$START;
              BASE,APTRADDR = ADDRPTR;
              CALL PRINT(.(' WHICH COLLIDES WITH $'));
              CALL PRINT$ID;
              CALL SETADDRPTR(0);
              CALL TAB2;
         END;
         IF ADDRPTR = 00H THEN
     CALL PRINT(.('AND THERE ARE NO FURTHER COLLISIONS $'));
         ELSE DO;
     CALL PRINT(.(' ANY OTHER COLLISIONS OCCUR IN THE$'));
           CALL CRLF;
           CALL PRINT(.('BUILT-IN SYMBOL TABLE $'));
           END;
         BASE = SAVE$BASE;
    END;
    CALL CRLF;
END CHECK$COLLISION;


ENTRY$HEAD: PROC;
    CALL WRITE$ENTRY;
    CALL PRINT$ID;
    CALL CHECK$COLLISION;
    CALL TAB1;
END ENTRY$HEAD;


CHECK$TYPE: PROC(A);
    DCL A BYTE;
    DCL TYPE BYTE;
    TYPE = (SHR(A, 3) AND FORMMASK);
    DO CASE TYPE;
         /* SCALAR-ORDINATE */
         CALL PRINT(.(' SCALAR ORDINATE $'));
         /* INTEGER */
         CALL PRINT(.(' INTEGER $'));
         /* CHARACTER */
         CALL PRINT(.(' CHARACTER $'));
         /* REAL */
         CALL PRINT(.(' REAL $'));
         /* COMPLEX */
         DO;
           SAVE$BASE = BASE;
           CALL SET$PAST$PN(9);
           BASE = ADDRPTR;
           CALL SETADDRPTR(4);
```

```
          IF (BYTEPTR AND FORM$MASK) = 07H THEN
            CALL USER$DEFINED;
          ELSE CALL PRINT$ID;
          BASE = SAVE$BASE;
        END;
        /* BOOLEAN */
        CALL PRINT(.('BOOLEAN $'));
    END;   /* CASE TYPE */
    CALL CRLF;
    CALL TAB1;
END CHECK$TYPE;


CHECK$TYPE$CONST: PROC(A);
/* CHECK FOR TYPE OF CONSTANT AND PRINT IT */
  DCL A BYTE;
  DO CASE A;
    /* 0 UNSIGNED IDENTIFIER */
    CALL PRINT(.(' UNSIGNED IDENTIFIER $'));
    /* 1 INTEGER */
    CALL PRINT(.(' INTEGER $'));
    /* 2 REAL */
    CALL PRINT(.(' REAL $'));
    /* 3 STRING */
    CALL PRINT(.(' STRING $'));
    /* 4,5,6,7 NOT DEFINED */
       ; ; ; ;
    /* 8 SIGNED IDENTIFIER */
    CALL PRINT(.(' SIGNED IDENTIFIER $'));
  END; /* CASE */
END CHECK$TYPE$CONST;



PRINT$PRT: PROC(A);
    DCL A BYTE;
    IF A = 12 THEN
CALL PRINT(.('THE ASSIGNED PRT LOCATION FOR THE SBP IS $'));
    ELSE CALL PRINT(.('THE ASSIGNED PRT LOCATION IS $'));
    CALL SET$PAST$PN(A);
    CALL PRINT$DEC(ADDRPTR);
    CALL CRLF;
END PRINT$PRT;


PRINT$LABEL: PROC;
    CALL ENTRY$HEAD;
    CALL PRINT(.('THE ASSIGNED LABEL VALUE IS $'));
    CALL SET$PAST$PN(7);
    CALL PRINT$DEC(ADDRPTR);
    CALL CRLF;
END PRINT$LABEL;
```

```
PRINT$CONST: PROC;
  DCL (TYPE,SIZE,I) BYTE;
    CALL WRITE$ENTRY;
    CALL PRINT$ID;
    CALL CHECK$COLLISION;
    CALL PRINT(.('      THE CONSTANT TYPE IS$'));
    TYPE = (SHR(FORM,3) AND 0FH);
    CALL CHECK$TYPE$CONST(TYPE);
    CALL CRLF;
    CALL PRINT(.('      THE CONSTANT VALUE = $'));
    IF TYPE = 1 THEN
      DO;
      CALL SET$PAST$PN(7);
      CALL PRINT$DEC(ADDRPTR);
      LPN=LPN+9;
      END;
    IF TYPE = 2 THEN
      DO;
      CALL SET$PAST$PN(7);
      CALL PRINT$BCD(8);
      LPN=LPN+15;
      END;
    IF (TYPE = 0) OR (TYPE = 3) OR (TYPE = 8) THEN
      DO;
      CALL SET$PAST$PN(7);
      SIZE = BYTEPTR;
      DO I = 1 TO SIZE;
        CALL SET$ADDR$PTR(7+LPN+I);
        CALL PRINT$CHAR(BYTEPTR);
        END;
      LPN=LPN+SIZE+8;
      END;
END PRINT$CONST;


PRINT$TYPE: PROC;
    CALL ENTRY$HEAD;
    CALL PRINT(.('THE PARENT TYPE IS $'));
    DO CASE (SHR(FORM,3) AND FORMMASK);
      CALL PRINT(.('INTEGER $')); /* 0 */
      CALL PRINT(.('REAL $')); /* 1 */
      CALL PRINT(.('CHAR $')); /* 2 */
      CALL PRINT(.('BOOLEAN $')); /* 3 */
      ; /* 4 */
      ; /* 5 */
      ; /* 6 */
      DO; /* 7 */
        CALL SET$PAST$PN(7);
        SAVE$BASE = BASE;
        BASE = ADDRPTR;
        CALL SETADDRPTR(4);
        IF (BYTEPTR AND FORMMASK) = 07H THEN
          CALL USER$DEFINED;
        ELSE CALL PRINT$ID;
```

```
        BASE = SAVE$BASE;
      END;
    END;  /* OF CASE */
END PRINT$TYPE;


PRINT$VARIABLE: PROC;
    CALL ENTRY$HEAD;
    CALL PRINT(.('THE VARIABLE TYPE IS $'));
    CALL CHECK$TYPE(FORM);
    CALL PRINT$PRT(7);
END PRINT$VARIABLE;


SUBROUTINE: PROC;
    DCL J BYTE;
    CALL PRINT(.('THERE ARE $'));
    CALL SET$PAST$PN(7);
    J = BYTEPTR;
    CALL PRINT$DEC(BYTEPTR);
    CALL PRINT(.(' PARAMETERS $'));
    CALL CRLF;
    CALL SET$PAST$PN(8);
    PARM$LISTING(SUBRTN:=SUBRTN+1), APTRADDR = ADDRPTR;
    PARM$NUM(SUBRTN) = J;
    DO I = 1 TO J;
         CALL TAB2;
         CALL PRINT(.('NO. $'));
         CALL PRINT$DEC(I);
         CALL TAB1;
         IF SHR(BYTEPTR,7) THEN
         DO;
        IF SHR(BYTEPTR,6) THEN CALL PRINT(.(' FUNCTION $'));
              ELSE CALL PRINT(.(' VAR $'));
         END;
   ELSE IF BYTEPTR = 4 THEN CALL PRINT(.(' PROCEDURE $'));
              ELSE CALL PRINT(.(' VALUE $'));
         CALL PRINT(.('PARAMETER OF TYPE $'));
         CALL CHECK$TYPE(FORM);
         APTRADDR = APTRADDR + 3;
    END; /* DO I */
    CALL PRINT$PRT(10);
    CALL PRINT$PRT(12);
    CALL TAB1;
   CALL PRINT(.('THE LABEL VALUE PRECEDING THE CODE IS $'));
    CALL SET$PAST$PN(14);
    CALL PRINT$DEC(ADDRPTR);
    CALL CRLF;
END SUBROUTINE;


BRANCH: PROC;
    SBTBL = SBTBL + (3 * PARM$NUM(SUBRTN));
    SUBRTN = SUBRTN - 1;
```

```
END BRANCH;


PRINT$PROC: PROC;
    CALL ENTRY$HEAD;
    CALL SUBROUTINE;
END PRINT$PROC;


PRINT$FUNC: PROC;
    CALL ENTRY$HEAD;
    CALL PRINT(.('THE FUNCTION TYPE IS $'));
    CALL SET$PAST$PN(16);
    FORM = BYTEPTR;
    CALL CHECK$TYPE(FORM);
    CALL SUBROUTINE;
END PRINT$FUNC;


PRINT$FILE: PROC;
    CALL ENTRY$HEAD;
END PRINT$FILE;


SKIPPER: PROC;
    DO CASE(SHR(FORM,3) AND FORMMASK);
      DO;
        CALL SETADDRPTR(6);
        SBTBL = SBTBL + 10 + BYTEPTR;
      END;
      SBTBL = SBTBL + 16;
      DO;
        CALL SETADDRPTR(5);
        SBTBL = SBTBL + 10 + (2 * BYTEPTR);
      END;
      DO;
        IF FORM = 1FH THEN SBTBL = SBTBL + 9;
        ELSE DO;
          CALL SETADDRPTR(6);
          SBTBL = SBTBL + 14 + BYTEPTR;
        END;
      END;
      SBTBL = SBTBL + 7;
      SBTBL = SBTBL + 7;
      SBTBL = SBTBL + 7;
    END;  /* OF CASE */
END SKIPPER;


STARS: PROC;
    CALL CRLF;
CALL
PRINT(.('---------------------------------------$'));
    CALL CRLF;
```

```
END;


MOVE: PROC(SOURCE,DESTIN,L);
    DCL (SOURCE,DESTIN,L) ADDR,
        (SCHAR BASED SOURCE, DCHAR BASED DESTIN) BYTE;
    DO WHILE (L:=L - 1) <> 65535;
       DCHAR=SCHAR;
       DESTIN=DESTIN+1;
       SOURCE=SOURCE+1;
    END;
END MOVE;


MAINLINE:DO;
CALL DOTSYM;

BASE, SBTBL = .MEMORY;
L = COPY$SBTBL;
CALL SETADDRPTR(4);
FORM = BYTEPTR;
DO CASE (FORM AND FORMMASK);
    CALL SET$PAST$PN(11);
    DO;
      CALL SETADDRPTR(4);
      IF SHR(BYTEPTR,4) THEN CALL SET$PAST$PN(17);
      ELSE CALL SET$PAST$PN(11);
    END;
    CALL SET$PAST$PN(11);
    CALL SET$PAST$PN(13);
    CALL SET$PAST$PN(18);
    CALL SET$PAST$PN(19);
    CALL SET$PAST$PN(9);
      /* THIS ENTRY IS IMPOSSIBLE FOR THE FIRST ENTRY */
END; /* CASE FORM */
/* STARTING LOCATION OF THE SYMBOL TABLE */
TABLE$START = ADDRPTR;
CALL MOVE(SBTBL,TABLE$START,L);
BASE,SBTBL = TABLE$START;
/* START */
CALL SETADDRPTR(2);
DO WHILE ADDRPTR <> 00H;
    CALL SETADDRPTR(4);
    FORM = BYTEPTR;
    CALL STARS;
    DO CASE (BYTEPTR AND FORMMASK);
    /* LABEL */
         DO;
              CALL PRINT$LABEL;
              SBTBL = SBTBL + 9 + LPN;
         END;
         /* CONSTANT */
         DO;
              CALL PRINT$CONST;
```

```
                SBTBL = SBTBL + LPN;
          END;
          /* TYPE */
          DO;
                CALL PRINT$TYPE;
                SBTBL = SBTBL + 9 + LPN;
          END;
          /* VARIABLE */
          DO;
                CALL PRINT$VARIABLE;
                SBTBL = SBTBL + 11 + LPN;
          END;
          /* PROCEDURE */
          DO;
                CALL PRINT$PROC;
                SBTBL = SBTBL + 16 + LPN;
          END;
          /* FUNCTION */
          DO;
                CALL PRINT$FUNC;
                SBTBL = SBTBL + 17 + LPN;
          END;
          /* FILE */
                DO;
                CALL PRINT$FILE;
                SBTBL = SBTBL + 7 + LPN;
          END;
          /* USER DEFINED ENTRY */
          DO;
            CALL SKIPPER;
          END;
    END; /* OF CASE */
    IF SBTBL = PARM$LISTING(SUBRTN) THEN CALL BRANCH;
    BASE = SBTBL;
    CALL SETADDRPTR(2);
END;
CALL CRLF;
CALL PRINT(.('THE SYMBOL TABLE HAS BEEN PRINTED. $'));
CALL BOOT;
END MAINLINE;
END SYM;
```

LIST OF REFERENCES

1. Gracida, J. C. and Stilwell, R.R., NPS-PASCAL: A Partial Implementation of PASCAL Language for a Microprocessor-based Computer System, Master Thesis, Naval Postgraduate School, Monterey, CA, June 1978.

2. Brynes, J. L., NPS-PASCAL A PASCAL Implementation for Microprocessor-based Computer Systems, Master Thesis, Naval Postgraduate School, Monterey, CA, June 1979.

3. Jensen, K., and Wirth, N., "Pascal User Manual and Report", 2nd ed., Springer-Verlag, New York - Heidelberg - Berlin, 1974.

4. Sale, A. H. J., "The Pascal Validation Suite Version 2.2", PASCAL NEWS, Number 16, p.11-153, October 1979.

5. British Standards Institute/International Standards Organization Working Draft/3 for Pascal Standard, PASCAL NEWS, Number 14, p.3-60, January 1979.

6. University of Toronto Computer Systems Research Group Technical Report CSRG-2, An Efficient LALR Parser Generator, by W. R. Lalonge, April, 1971.

7. Flynn, J. P. and Moranville, M. S., ALGOL-M An Implementation Of A High Level Block Structured Language For A Microprocessor-based Computer System, Naval Postgraduate School, Monterey, CA, September 1977.

8. Intel Corporation, 8080/8085 Assembly Language Programming Manual, 1978, 3065 Bowers Ave., Santa Clara, CA, 95051.

9. Blanton, G. B. and Moore, E. F., Implementation of Subroutines on the NPS-PASCAL Compiler, paper submitted for CS3113, Introduction to Compilers, Naval Postgraduate School, Monterey, CA, 2 October 1979.

10. Naval Postgraduate School Report NPS-53KI72011A, ALGOL-E: An Experimental Approach to the Study of Programming Langauges, by G. A. Kildall, 7 January 1972.

11. Anderson, R. and Myers, L., Implementation of Record Structures for the NPS-PASCAL Compiler, paper submitted for CS3113, Introduction to Compilers, Naval Postgraduate School, Monterey, CA, 12 December 1979.

12. Intel Corporation, ISIS-II User's Guide, 1978.

13. Intel Corporation, ISIS-II PL/M-80 Compiler Operator's Manual, 1977.

14. Intel Corporation, PL/M-80 Programming Manual, 1978.

15. Digital Research, An Introduction to CP/M Features and Facilities, 1978, Box 579, Pacific Grove, CA, 93950.

16. Digital Research, SID User's Manual, 1978.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
4.	Asst Professor B. J. MacLennan, Code 52Ml Department of Computer Science Naval Postgraduate School Monterey. California 93940	1
5.	LT Mark S. Moranville, USN, Code 52Mi Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
6.	LCDR Robert R. Stilwell, USN, Code 52Sb Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
7.	LCDR Frank Burkhead, USN, Code 52Bg Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
8.	Microcomputer Laboratory, Code 0131 Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
9.	Capt Konrad S. Tinius, USAF HQ USEUCOM (ECADP) APO NY 09128	1